Consensus-Halving: Does It Ever Get Easier?

Aris Filos-Ratsikas
University of Liverpool, United Kingdom
Aris.Filos-Ratsikas@liverpool.ac.uk

Alexandros Hollender
University of Oxford, United Kingdom
Alexandros.Hollender@cs.ox.ac.uk

Katerina Sotiraki
Massachusetts Institute of Technology, USA
katesot@mit.edu

Manolis Zampetakis
Massachusetts Institute of Technology, USA
mzampet@mit.edu

March 2, 2020

Abstract

In the ε-*Consensus-Halving* problem, a fundamental problem in fair division, there are n agents with valuations over the interval $[0, 1]$, and the goal is to divide the interval into pieces and assign a label "$+$" or "$-$" to each piece, such that every agent values the total amount of "$+$" and the total amount of "$-$" almost equally. The problem was recently proven by Filos-Ratsikas and Goldberg [2018, 2019] to be the first "natural" complete problem for the computational class PPA, answering a decade-old open question.

In this paper, we examine the extent to which the problem becomes easy to solve, if one restricts the class of valuation functions. To this end, we provide the following contributions. First, we obtain a strengthening of the PPA-hardness result of [Filos-Ratsikas and Goldberg, 2019], to the case when agents have *piecewise uniform* valuations with only *two blocks*. We obtain this result via a new reduction, which is in fact conceptually much simpler than the corresponding one in [Filos-Ratsikas and Goldberg, 2019]. Then, we consider the case of *single-block (uniform) valuations* and provide a parameterized polynomial time algorithm for solving ε-*Consensus-Halving* for any ε, as well as a polynomial-time algorithm for $\varepsilon = 1/2$; these are the first algorithmic results for the problem. Finally, an important application of our new techniques is the first hardness result for a generalization of Consensus-Halving, the Consensus-$1/k$-Division problem [Simmons and Su, 2003]. In particular, we prove that ε-Consensus-$1/3$-Division is PPAD-hard.

1 Introduction

The topic of *fair division* has been in the focus of research in economics and mathematics, since the late 1940s and the pioneering works of Banach, Knaster and Steinhaus [Steinhaus, 1948], who developed the associated theory. The related literature contains many interesting problems, with the most celebrated perhaps being the problems of *envy-free cake-cutting* and *equitable cake-cutting*, for which a plethora of results have been obtained. More recently, the computer science literature has made a significant contribution in studying the computational complexity of these problems, and attempting to design efficient algorithms for several of their variants [Aziz and Mackenzie, 2016a,b; Deng et al., 2012; Arunachaleswaran et al., 2019].

Another classical problem in fair division, whose study originates back to as early as the 1940s

and the work of Neyman [1946], is the *Consensus-Halving problem*[1] [Simmons and Su, 2003]. In this problem, there is a set of n agents with valuation functions over the $I = [0,1]$ interval. The goal is to divide the interval into pieces using at most n cuts, and assign a label from $\{+,-\}$ to each piece, such that every agent values the total amount of I labeled "$+$" and the total amount of I labeled "$-$" equally. Similarly to other well-known problems in fair division, the existence of a solution to the Consensus-Halving problem is *always* guaranteed, and can be proven via the application of a fixed-point theorem; here the *Borsuk-Ulam theorem* [Borsuk, 1933]. As a matter of fact, the problem is a continuous analogue of the well-known Necklace Splitting problem [Goldberg and West, 1985; Alon, 1987], whose existence of a solution is typically established via an existence proof for the continuous version.

The Consensus-Halving problem attracted attention in the literature of computer science recently, due to the breakthrough results of Filos-Ratsikas and Goldberg [2018, 2019] who studied the computational complexity of the approximate version, in which there is a small allowable discrepancy ε between the values of the two portions. First, in [Filos-Ratsikas and Goldberg, 2018], the authors proved that ε-Consensus-Halving for inverse-exponential ε is complete for the computational class PPA, defined by Papadimitriou [1994]. This was the first PPA-completeness result for a "natural" problem, i.e., a computational problem that does not have a polynomial-sized circuit explicitly in its definition, answering an open question from Papadimitriou [1994], reiterated multiple times over the years [Grigni, 2001; Aisenberg et al., 2020]. Then in [Filos-Ratsikas and Goldberg, 2019], the authors strengthened their hardness result to the case of inverse-polynomial ε, which also established the PPA-completeness of the Necklace Splitting problem for 2 thieves.

Despite the aforementioned results, the complexity of the problem is not yet well understood. Does the problem remain hard if one restricts attention to classes of simple valuation functions? Note that the reduction of [Filos-Ratsikas and Goldberg, 2018, 2019] uses instances with *piecewise constant* valuation functions with *polynomially many* pieces. On the opposite side, are there efficient algorithms for solving special cases of the problem? What if we allow a larger number of cuts?

1.1 Our Results

Towards understanding the complexity of Consensus-Halving, we present the following results.

- We prove that ε-Consensus-Halving is PPA-complete, even when the agents have *two-block uniform* valuations, i.e., valuation functions which are *piecewise uniform* over the interval and assign non-zero value on at most *two* pieces. This result holds even when ε is inverse-polynomial, and extends to the case where the number of allowable cuts is $n + n^{1-\delta}$, for some constant $\delta > 0$.

  This is an important strengthening of the results in [Filos-Ratsikas and Goldberg, 2018, 2019] which require the agents to have piecewise constant valuations with polynomially many non-uniform blocks. En route to this result, we obtain a significant simplification to the proof of [Filos-Ratsikas and Goldberg, 2018, 2019], which uses new gadgets for the encoding of the circuit of high-dimensional Tucker (see Definition 6), which we reduce from. Our new reduction also gives a much simplified proof of PPA-completeness for Necklace Splitting with 2 thieves.

[1] The name "Consensus-Halving" is attributed to Simmons and Su [2003], although the problem has been studied under different names in the past. For example, it is also known as *The Hobby-Rice theorem* [Hobby and Rice, 1965], or *continuous necklace splitting* [Alon, 1987].

- We study the case of *single-block valuations* and provide the first algorithmic results for the problem.[2] Specifically, we present:

  - an algorithm for any ε, whose running time is parameterized by the maximum number d of intersections between blocks and is polynomial in $1/\varepsilon$ (i.e., an FPTAS when d is constant),
  - a polynomial-time algorithm for 1/2-Consensus-Halving.

  We complement our main results with a simple algorithm based on linear programming, which solves the problem for single-block valuations in polynomial-time, if one is allowed to use $2n - \ell$ cuts, for any constant ℓ.

- As an application of the new ideas developed in our reduction, we obtain the first hardness result for a generalization of ε-Consensus-Halving, known as ε-*Consensus*-1/k-*Division*, for $k \geq 3$. Specifically, we prove that ε-*Consensus*-1/3-*Division* is PPAD-hard, when ε is inverse-exponential.

1.2 Discussion and Related Work

The study of the Consensus-Halving problem originates back to the early 1940s and the work of Neyman [1946]. The first proof of existence for n cuts can be traced back to the 1965 theorem of Hobby and Rice [Hobby and Rice, 1965]. The problem was famously studied in the context of Necklace Splitting, being a continuous analogue of the latter problem; in fact, most known proofs for Necklace Splitting go via the continuous version[3] [Goldberg and West, 1985; Alon and West, 1986]. The name *Consensus-Halving* is attributed to Simmons and Su [2003], who studied the continuous problem independently, and came up with a constructive proof of existence. Their construction, although yielding an exponential-time algorithm, was later adapted by Filos-Ratsikas et al. [2018] to prove that the problem lies in the computational class PPA.

The class PPA was defined by Papadimitriou [1994] in his seminal paper in 1994, in which he also defined several other important subclasses of TFNP [Megiddo and Papadimitriou, 1991], the class of *Total Search Problems in NP*, i.e., problems that *always* have solutions which are efficiently verifiable. Among those classes, the class PPAD has been very successful in capturing the complexity of many interesting computational problems [Mehta, 2014; Garg et al., 2018; Goldberg and Hollender, 2019; Chen et al., 2013], highlighted by the celebrated result of Daskalakis et al. [2009] and Chen et al. [2009] about the PPAD-completeness of computing a Nash equilibrium. On the contrary, since the definition of the class, PPA was not known to contain any natural complete problems, but rather mostly versions of PPAD-complete problems of a topological nature, defined on non-orientable spaces [Deng et al., 2016; Grigni, 2001]. In 2015, Aisenberg et al. [2020] showed that the computational version of Tucker's Lemma [Tucker, 1945], already shown to be in PPA by Papadimitriou [1994], is actually complete for the class.

Using the latter result as a starting point, Filos-Ratsikas and Goldberg [2018] proved that ε-Consensus-Halving is PPA-complete when ε is inverse exponential. This was a breakthrough result in the following sense: it was the first PPA-completeness result for a "natural" computational

[2] To be precise, we provide the first such results for the version of the problem with n agents and n cuts. For a large number of cuts, Brams and Taylor [1996] present algorithms for ε-approximate solutions. Additionally, these algorithms require a number of cuts which grows as ε decreases, while our results for more than n cuts are not dependent on ε.

[3] This is true for the case of 2 thieves. For k thieves, the proofs go via the Consensus-1/k-Division problem instead.

problem, where the term "natural" takes the specific meaning of a problem that does not have a polynomial-sized circuit in its definition. The quest for such problems that would be complete for PPA was initiated by Papadimitriou himself [Papadimitriou, 1994] and was later brought up again by several authors, including Grigni [2001] and Aisenberg et al. [2020]. In the same paper, the authors also provided a computational equivalence between the ε-Consensus-Halving problem and the well-known Necklace Splitting problem of Alon [1987] for 2 thieves [Goldberg and West, 1985; Alon and West, 1986], when ε is inverse-polynomial. In [Filos-Ratsikas and Goldberg, 2019], the authors strengthened their result to ε being inverse-polynomial, which, together with the aforementioned result from [Filos-Ratsikas and Goldberg, 2018], also provided a proof for the PPA-completeness of Necklace Splitting. As we mentioned earlier, besides being a strengthening, our PPA-hardness proof for ε-Consensus-Halving is a significant simplification over that of [Filos-Ratsikas and Goldberg, 2019], and importantly, it holds for ε which is inverse-polynomial. Therefore, we also obtain a new, simplified proof of PPA-hardness for Necklace Splitting with 2 thieves.

For constant ε, the only hardness result that we know is the PPAD-hardness of Filos-Ratsikas et al. [2018], who also show that when $n-1$ cuts are allowed, deciding whether a solution exists is NP-hard. Recently, Deligkas et al. [2019] studied the complexity of *exact* Consensus-Halving and showed that the problem is FIXP-hard. Interestingly, the authors also introduced a new computational class, called BU (for Borsuk-Ulam) and showed that the problem lies in that class, leaving open the question of whether it is BU-complete.

If we generalize the number of labels to $\{1, 2, \ldots, k\}$ rather that $\{+, -\}$, and we allow $(k-1)n$ cuts rather than only n, then we obtain a generalization of the Consensus-Halving problem which was referred to as *Consensus-1/k-Division* in [Simmons and Su, 2003]. The existence of a solution for this problem can be proved via fixed-point theorems that generalize the Borsuk-Ulam theorem [Bárány et al., 1981; Alon, 1987], however very little is known about its complexity. One might feel inclined to believe that Consensus-1/k-Division is a harder problem that Consensus-Halving; however, note that in the former problem, we have more cuts at our disposal. In fact, Filos-Ratsikas and Goldberg [2019] conjectured that the complexities of the problems for different values of k are incomparable, and are characterized by different complexity classes. The complexity classes that are believed to be the most related are called PPA-k, defined also by Papadimitriou [1994] in his original paper; we refer the reader to the recent papers of [Göös et al., 2019; Hollender, 2019] for a more detailed discussion of these classes.

Before our paper, virtually nothing was known about the hardness of the problem when $k \geq 3$. While the techniques in [Filos-Ratsikas and Goldberg, 2019] were highly reliant on the presence of only two labels, our ideas do carry over to the case when $k = 3$, which enables us to prove our PPAD-hardness result. While we do not expect the problem for $k \geq 3$ to be PPAD-complete, our proof offers important intuition about the intricacies of the problem and could be useful for proving stronger hardness results in the future.

2 Preliminaries

We start with the definition of the ε-approximate version of the Consensus-Halving problem.

Definition 1 (ε-Consensus-Halving)**.** Let $k \geq 2$. We are given $\varepsilon > 0$ and a set $\mathcal{C}$ of continuous probability measures $\mu_1, \ldots, \mu_n$ on $I = [0, 1]$. The probability measures are given by their density functions on I. The goal is to partition the unit interval into 2 (not necessarily connected) pieces I^+ and I^- using at most n cuts, such that $|\mu_j(I^+) - \mu_j(I^-)| \leq \varepsilon$ for all agents $j \in \{1, \ldots, n\}$.

We will refer to the probability measures $\mu_1, \ldots, \mu_n$ as *valuation functions* or simply *valuations*. While the existence and PPA-membership results hold more generally, in this paper, we will restrict our attention to the case when the valuation functions are *piecewise constant*. These can be represented explicitly in the input as endpoints and heights of value blocks.

Definition 2 (Piecewise constant valuation functions)**.** A valuation function μ_i is *piecewise constant* over an interval I, if the domain can be partitioned into a finite set of intervals such that the density of μ_i is constant over each interval.

Piecewise constant functions are often referred to as *step functions*.

Definition 3 (Uniform valuation functions)**.** We will consider the following subclasses of piecewise constant valuation functions.

- ***Piecewise Uniform:***, The domain can be partitioned into a finite set of intervals such that the density of μ_i is either v_i or 0 over each interval, for some constant v_i.
- ***d*-block Uniform:** The domain can be partitioned into a finite set of intervals, such that in at most d of those the density of μ_i is v_i and everywhere else it is 0, for some constant v_i.
- **2-*block Uniform:*** d-block uniform valuations for $d = 2$.
- ***Single-block:*** d-block uniform valuations for $d = 1$. Here we omit the term "uniform", as there is only a single value block.

Obviously, piecewise constant $\supseteq$ piecewise uniform $\supseteq$ 2-block uniform $\supseteq$ single-block.

2.1 The Computational Classes PPA and PPAD

As we mentioned in the introduction, Consensus-Halving is a *Total Search Problem in NP*, i.e., a problem with a guaranteed solution which is verifiable in polynomial time. The corresponding class is the class TFNP [Megiddo and Papadimitriou, 1991]. Formally, a binary relation $P(x, y)$ is in the class TFNP if for every x, there exists a y of size bounded by a polynomial in $|x|$ such that $P(x, y)$ holds and $P(x, y)$ can be verified in polynomial time. The problem is given x, to find such a y in polynomial time.

The subclasses of TFNP that will be relevant for this paper are PPAD and PPA [Papadimitriou, 1994]. These are defined via their canonical problems, End-of-Line and Leaf respectively.

Definition 4 (End-of-Line)**.** The input to the End-of-Line problem consists of two Boolean circuits S (for successor) and P (for predecessor) with n inputs and n outputs such that $P(0^n) = 0^n \neq S(0^n)$, and the goal is to find a vertex x such that $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^n$.

A problem is *in* PPAD if it is polynomial-time reducible to End-Of-Line and it is PPAD-*complete* if End-Of-Line reduces to it in polynomial-time. Intuitively, PPAD is defined with respect to a directed graph of exponential size, which is given *implicitly* as input, via the use of the predecessor and successor circuits defined above. PPAD is a subclass of PPA, which is defined similarly, but with respect to an undirected graph and a circuit that outputs the neighbours of a vertex. Its canonical computational problem is called Leaf, which is defined below.

Definition 5 (Leaf)**.** The input to the Leaf problem is a Boolean circuit C with n inputs and at most $2n$ outputs, outputting the set $\mathcal{N}(y)$ of (at most two) neighbors of a vertex y, such that $|\mathcal{N}(0^n)| = 1$, and the goal is to find a vertex x such that $x \neq 0^n$ and $|\mathcal{N}(x)| = 1$.

A problem is the class PPA if it is polynomial-time reducible to Leaf and is PPA-complete if Leaf reduces to it in polynomial time.

2.2 High-dimensional Tucker

Our reduction in Section 3 will start from the following problem, which is an N-dimensional variant of the 2D-Tucker problem [Papadimitriou, 1994; Aisenberg et al., 2020].

Definition 6 (high-D-Tucker)**.** An instance of high-D-Tucker consists of a labeling $\lambda : [8]^N \to \{\pm 1, \ldots, \pm N\}$ computed by a Boolean circuit. We further assume that the labeling is antipodally anti-symmetric (i.e. for all x on the boundary of $[8]^N$ it holds that $\lambda(\overline{x}) = -\lambda(x)$ where $\overline{x}_i = 9 - x_i$ for all i), which can be enforced syntactically. A solution consists of two points $x, y \in [8]^N$ with $\lambda(x) = -\lambda(y)$ and $\|x - y\|_\infty \leq 1$.

Filos-Ratsikas and Goldberg [2019] showed that the problem is PPA-hard, when the domain is $[7]^N$ instead of $[8]^N$. We adapt the hardness to the case of Definition 6 in the theorem below.

Theorem 1. high-D-Tucker *is* PPA*-complete.*

Proof. Papadimitriou [1994] has shown that the problem lies in PPA. In order to show PPA-hardness, we use the fact that Filos-Ratsikas and Goldberg [2019] have proved that the problem is PPA-hard on the domain $[7]^N$ (instead of $[8]^N$) by using a standard snake-embedding technique Chen et al. [2009]; Deng et al. [2017].

Let λ be an instance of High-D-Tucker but on the domain $[7]^N$ instead of $[8]^N$. We will reduce this to an instance λ' of High-D-Tucker (on our standard domain $[8]^N$). In the two-dimensional case ($N = 2$), the high level idea of the reduction is to take the domain $[7]^N$ and duplicate the central vertical and horizontal lines of the grid (thus also duplicating the labels at these grid points).

Formally, we proceed as follows. Define the operator $\widehat{\cdot}$ such that for any $r \in [8]$

$$\widehat{r} := \begin{cases} r - 1 & \text{if } r \geq 5 \\ r & \text{if } r \leq 4 \end{cases}$$

Then, for any $x = (x_1, \ldots, x_N) \in [8]^N$, let $\widehat{x} = (\widehat{x}_1, \ldots, \widehat{x}_N) \in [7]^N$. Now define λ' such that for all $x \in [8]^N$, $\lambda'(x) := \lambda(\widehat{x})$. This is well-defined and given a circuit that computes λ, we can construct a circuit for λ' in polynomial time.

Let us first show that if λ is antipodally anti-symmetric on $[7]^N$, then λ' is antipodally anti-symmetric on $[8]^N$. Consider any $x \in \partial([8]^N)$, i.e. there exists $j \in [N]$ such that $x_j \in \{1, 8\}$. Note that we then have $\widehat{x} \in \partial([7]^N)$, because $\widehat{x}_j \in \{1, 7\}$. Thus, we know that $\lambda(8 - \widehat{x}_1, \ldots, 8 - \widehat{x}_N) = -\lambda(\widehat{x}_1, \ldots, \widehat{x}_N)$. Using the key observation that $\widehat{9 - x_i} = 8 - \widehat{x}_i$ for all $i \in [N]$, we obtain that

$$\lambda'(9 - x_1, \ldots, 9 - x_N) = \lambda(8 - \widehat{x}_1, \ldots, 8 - \widehat{x}_N) = -\lambda(\widehat{x}_1, \ldots, \widehat{x}_N) = -\lambda'(x_1, \ldots, x_N).$$

It remains to show that given any solution to λ', we can retrieve in polynomial time a solution to λ. Let $x, y \in [8]^N$ be such that $\lambda'(x) = -\lambda'(y)$ and $\|x - y\|_\infty \leq 1$. Then, we immediately obtain that $\lambda(\widehat{x}) = -\lambda(\widehat{y})$ and it remains to show that $\|\widehat{x} - \widehat{y}\|_\infty \leq 1$. Consider any $i \in [N]$. If $x_i, y_i \geq 5$ or if $x_i, y_i \leq 4$, then in both cases $|x_i - y_i| \leq 1$ implies $|\widehat{x}_i - \widehat{y}_i| \leq 1$. If $x_i \geq 5$ and $y_i \leq 4$, then $|x_i - y_i| \leq 1$ implies that $x_i = 5$ and $y_i = 4$ and we get $|\widehat{x}_i - \widehat{y}_i| = 0 \leq 1$. The remaining case is analogous. Thus, we have shown that $\widehat{x}, \widehat{y}$ form a solution to λ. □

3 Consensus-Halving with two-block uniform valuations is PPA-hard

In this section, we present our first result, regarding the PPA-hardness of Consensus-Halving.

Theorem 2. *ε-Consensus-Halving is PPA-hard, when ε is inverse-polynomial and the agents have two-block uniform valuations.*

As we mentioned in the Introduction, Theorem 2 is a strengthening of the result of Filos-Ratsikas and Goldberg [2019], which requires the valuation functions to have a polynomial number of value blocks, and which is seemingly very difficult to extend to two-block uniform valuations. To achieve this stronger result, we have to develop new gadgetry, based on a new interpretation of the cut positions with respect to the positions of points in the domain of high-D-Tucker. As it turns out, this new interpretation allows us to obtain a new proof of the main theorem of Filos-Ratsikas and Goldberg [2019], one which is conceptually much simpler, even though it actually applies to more restricted valuations.

Before we proceed, we first remark the following. In Filos-Ratsikas et al. [2018] (where the PPAD-hardness of ε-Consensus-Halving was proven for constant ε) the authors presented a simple argument that allowed them to extend their hardness result to $n + c$ cuts, where c is some constant. The idea is to make $c + 1$ *completely disjoint* copies of the instance of ε-Consensus-Halving, and solve it using $n + c$ cuts. One of the copies would have to be solved using at most n cuts, which is a PPAD-hard problem. We observe that the same principle applies generically (beyond PPAD-hardness and also to the results of Filos-Ratsikas and Goldberg [2018, 2019]), and in fact extends to $n + n^{1-\delta}$ cuts, where $\delta > 0$ is some constant. From Theorem 2, we obtain the following corollary.

Corollary 3. *ε-Consensus-Halving is PPA-hard, when ε is inverse-polynomial and the agents have two-block uniform valuations, even when one is allowed to use $n + n^{1-\delta}$ cuts, for constant $\delta > 0$.*

We are now ready to prove Theorem 2. We first provide an overview of the reduction and we highlight the main simplifications over the proof of Filos-Ratsikas and Goldberg [2019]. Then we proceed to formally present the proof of Theorem 2.

3.1 Overview of the reduction

We are given an instance of high-D-Tucker, namely a labeling $\lambda : [8]^N \to \{\pm 1, \ldots, \pm N\}$ computed by a Boolean circuit. We will show how to construct an instance of Consensus-Halving in polynomial time such that any ε-approximate solution yields a solution to the high-D-Tucker instance (for some inversely-polynomial ε). The complexity will be measured with respect to the representation size of the high-D-Tucker instance, i.e., the size of the circuit λ (which is also at least N).

For clarity and convenience, the instance of Consensus-Halving we will construct will not be defined on the domain $[0, 1]$, but instead on some interval $[0, M]$, where M is bounded by a polynomial in the size of the high-D-Tucker circuit λ. It is easy to transform this into an instance on $[0, 1]$ by just re-scaling the valuation functions, namely scaling down the positions of the blocks by M and scaling up the heights of the blocks by M.

Overview. Let us first provide a very high-level description of the instance we construct. Similarly to Filos-Ratsikas and Goldberg [2019], the left-most end of the instance will be the *Coordinate-Encoding* region. In any solution S to the instance, the way in which this region is divided amongst the labels $+$ and $-$ will represent a point $x \in [-1, 1]^N$. A *circuit-simulator* C will read-in the coordinates of x, perform some computations (including a simulation of λ) and output N values $[C(x)]_1, \ldots, [C(x)]_N \in [-1, 1]$. The circuit-simulator will consist of a set of agents and each agent will implement one gate/operation of the circuit. Unfortunately, the circuit-simulator

can sometimes fail to perform the desired computation, so instead of one circuit-simulator C we will actually have a polynomial number $p(N)$ of circuit-simulators $C_1, \ldots, C_{p(N)}$. Each of these circuit-simulators will be performing (almost) the same computation. Finally, we will introduce a *Feedback region* where N feedback agents $f_1, \ldots, f_N$ will implement the *feedback mechanism*. For each $i \in \{1, \ldots, N\}$, feedback agent f_i will ensure that $\frac{1}{p(N)} \sum_{j=1}^{p(N)} [C_j(x)]_i \approx 0$. Namely, it will ensure that the average of the outputs in dimension i is close to zero. We will show that from any solution S to the CONSENSUS-HALVING instance, we obtain a solution to the original HIGH-D-TUCKER instance.

Encoding of a value in $[-1, 1]$. Given any solution S of our instance, every interval I of length 1 of the domain encodes a value in $[-1, 1]$ as follows. Let I^+ and I^- denote the subsets of I labeled respectively $+$ and $-$ in the solution S. Then the value encoded by I, $v_S(I)$, is given by $\mu(I^+) - \mu(I^-)$, where μ is the Lebesgue measure on $\mathbb{R}$. Since there are at most n cuts (where n is the number of agents in the instance), I^+ is the union of at most $n+1$ disjoint sub-intervals of I and $\mu(I^+)$ is simply the sum of the lengths of these intervals (and the same holds for I^-). It is easy to see that $v_S(I) = 0$ corresponds to I being perfectly shared between $+$ and $-$ in S, whereas $v_S(I) = +1$ corresponds to the whole interval I being labeled $+$. We will drop the subscript S and just use $v(I)$ in the remainder of this exposition.

Coordinate-Encoding region. The sub-interval $[0, N]$ of the domain is called the *Coordinate-Encoding* region. Indeed, the way in which this region is subdivided amongst the $+$ and $-$ labels in a solution S will encode the coordinates of a point in $x \in [-1, 1]^N$. In more detail, $x_1 \in [-1, 1]$ will be given by $v([0, 1])$, i.e., the value encoded by interval $[0, 1]$. Similarly, $x_2 \in [-1, 1]$ will be given by $v([1, 2])$, $x_3 \in [-1, 1]$ by $v([2, 3])$, etc.

Constant-Creation region. The sub-interval $[N, N + p(N)]$ of the domain is called the *Constant-Creation* region. This region will be used to create the constants that the circuit-simulators need. The circuit-simulator C_1 will read-in the value $v([N, N+1]) =: \text{const}_1$ and will assume that it corresponds to the value $+1$. Note that given the constant $+1$, the circuit-simulator can create any constant $\zeta \in [-1, 1]$ by using a $\times\zeta$-gate (multiplication by the constant ζ). Similarly, the circuit-simulator C_2 will read-in the value $v([N+1, N+2]) =: \text{const}_2$ and use it as the constant $+1$, and so on for $C_3, C_4, \ldots, C_{p(N)}$.

If S is a solution such that the Constant-Creation region does not contain any cut, then the whole region will have the same label, and without loss of generality we can assume that this label is $+$. Thus, in such a solution S, all the circuit-simulators will indeed read-in the constant $+1$ from the Constant-Creation region, i.e., we will indeed have $\text{const}_j = +1$ for all $j = 1, \ldots, p(N)$.

Circuit-Simulation regions. For each $j \in \{1, 2, \ldots, p(N)\}$, the sub-interval $[N + p(N) + (j-1)q, N + p(N) + jq]$ of the domain will be used by the circuit-simulator C_j. The length q used by every circuit-simulator will be upper-bounded by some polynomial in N and the size of the circuit λ. Every circuit-simulator C_j will read-in the coordinates $x_1, \ldots, x_N \in [-1, 1]$ of the point x from the Coordinate-Encoding region, as well as the value $\text{const}_j \in [-1, 1]$ from the Constant-Creation region (and assume that it corresponds to the constant $+1$). Using these values, C_j will perform some computations, including a simulation of the Boolean circuit λ, and finally output N values $[C_j(x, \text{const}_j)]_1, \ldots, [C_j(x, \text{const}_j)]_N \in [-1, 1]$ into the *Feedback region*.

Feedback region. The Feedback region is located at the right end of the domain and is subdivided into N intervals $F_1, \ldots, F_N$ of length $p(N)$ each. For every $j \in [p(N)]$, let $F_i(j)$ denote the jth sub-interval of length 1 of F_i. The ith output of circuit-simulator C_j will be located in sub-interval $F_i(j)$. In other words, $v(F_i(j)) = [C_j(x, \text{const}_j)]_i$.

Every interval F_i will have a corresponding *feedback agent* f_i, who will ensure that the average of all the outputs in interval F_i is close to zero. In more detail, agent f_i will have a single block of value that covers interval F_i. As a result, this agent will be satisfied only if $\frac{1}{p(N)} \sum_{j=1}^{p(N)} v(F_i(j)) \in [-\varepsilon, \varepsilon]$.

Stray Cuts. Any agent belonging to a circuit-simulator performs a gate-operation. In Section 3.3, we introduce the different types of gates and how they are implemented by agents. One important feature of the agents implementing the gates is that every such agent ensures that at least one cut must lie in a specific interval J of the domain (in any solution S). By construction, we will make sure that these intervals are pairwise disjoint for different agents. Thus, every agent introduced as part of a circuit-simulator will force one cut to lie in a specific interval.

The only agents that are not part of a circuit-simulator are the feedback agents $f_1, \ldots, f_N$. Since the number of cuts in any solution is at most the number of agents, there are at most N cuts that are not constrained to lie in some specific interval. We call these the *free cuts*. The free cuts can theoretically "go" anywhere in the domain and interfere with the correct functioning of the circuit-simulators or the Constant-Creation region. The expected behavior of these N free cuts is that they should lie in the Coordinate-Encoding region. As such, any of the free cuts that lies outside the Coordinate-Encoding region will be called a *stray cut* (following Filos-Ratsikas and Goldberg [2019]).

Observation 1. *If there is at least one stray cut, then the point $x \in [-, 1, 1]^N$ encoded by the Coordinate-Encoding region lies on the boundary of $[-1, 1]^N$ (i.e., there exists i such that $|x_i| = 1$).*

Stray Cut interference. There are two ways for a stray cut to cause trouble:

1. it can *corrupt* a circuit, i.e., interfere with the correct functioning of the gates of a circuit-simulator. If the cut lies in the region of circuit-simulator C_j, then it can make a gate output the wrong result (i.e., not perform the desired operation). If the cut lies in the Constant-Creation region and intersects the interval that is used by circuit-simulator C_i to read-in the constant const_j, then it can have an effect such that $|\text{const}_j| \neq 1$. However, in any case, a single stray cut can only interfere with one circuit-simulator in this way. Thus, at most N circuit-simulators can suffer from this kind of interference. We will choose $p(N)$ large enough so that these corrupted circuit-simulators have a very limited influence.

2. it can interfere with the sign of const_j for many circuit-simulators C_j. Indeed, even a single stray cut can ensure that half of our circuit-simulators read-in the constant $+1$ and the other half read-in the constant -1. We will show that this is actually not a problem, and that it does not produce bogus solutions. Since stray cuts can only occur when x lies on the boundary of $[-1, 1]^N$ (Observation 1), the Tucker boundary conditions will be important for this.

Stray cuts that end up in the Feedback region do not have any effect. Indeed, the feedback agents $f_1, \ldots, f_N$ are immune to stray cuts. They always ensure that the average of the outputs is close to zero. Thus, a stray cut can only influence the outputs that a feedback agent sees (as detailed above), but not its functionality.

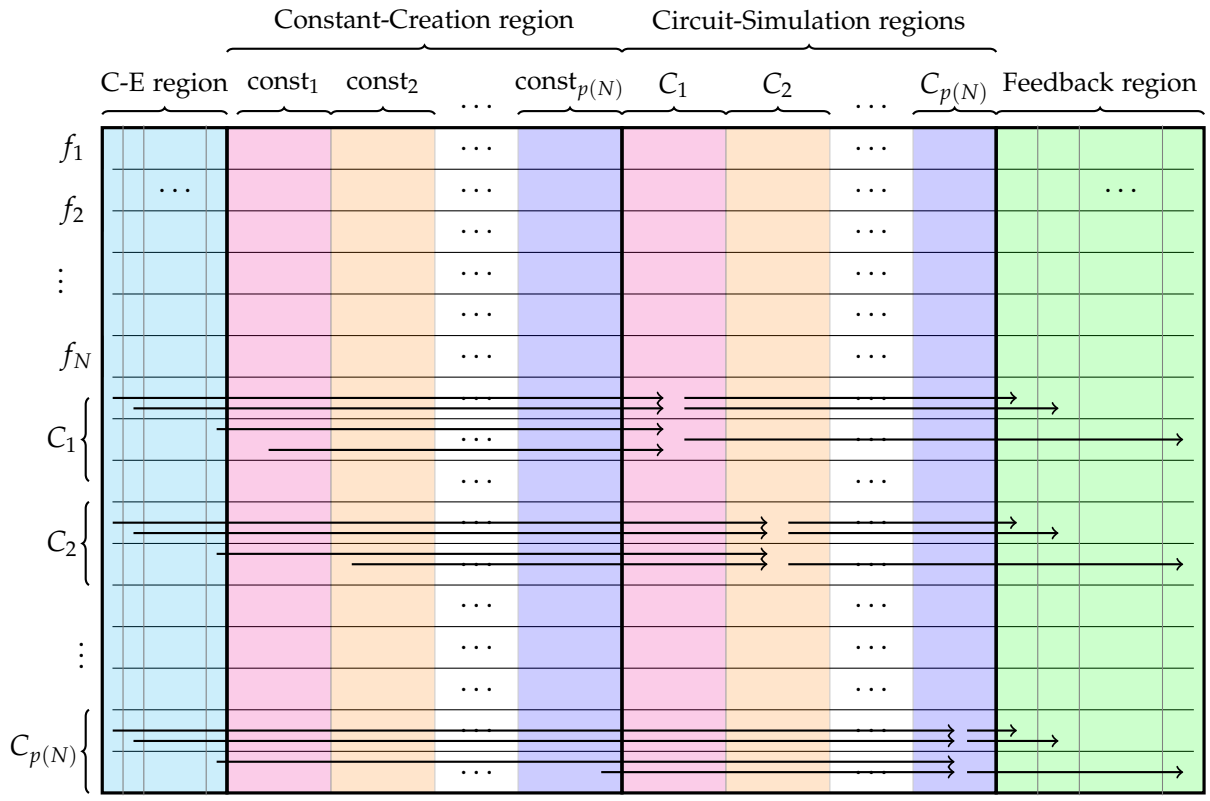

Figure 1: An overview of the different regions defined in the reduction. The regions corresponding to different Circuit-Simulators are color-coded. An arrow indicates that the region where it is pointing receives inputs from the region from where it is originating. On the left, the different types of agents are shown, namely the feedback agents, as well as the agents corresponding to the different Circuit-Simulators. The Coordinate-Encoding region and the Feedback Region are divided into sub-intervals, indicated by vertical gray lines, as detailed in Section 3.1.

Circuit-Simulator failure. There are two ways in which a circuit-simulator can fail to have the desired output:

1. it is corrupted by a stray cut. This can happen to at most N circuit-simulators.
2. it can fail in extracting the binary bits from (a point close to) x. We will ensure that this can happen to at most N circuit-simulators.

Thus, at most $2N$ circuit-simulators fail, i.e., at least $p(N) - 2N$ circuit-simulators have the desired output.

3.2 Major simplifications compared to [Filos-Ratsikas and Goldberg, 2019]

A much cleaner domain. The PPA-hardness of HIGH-D-TUCKER was already established in [Filos-Ratsikas and Goldberg, 2019] and our version can be obtained from that one using minor modifications, see Theorem 1. The corresponding result of [Filos-Ratsikas and Goldberg, 2019] is a standard application of the "snake-embedding" technique developed in [Chen et al., 2009]. However, the reduction in [Filos-Ratsikas and Goldberg, 2019] requires (a) a further constraint

on how the domain is colored and more importantly (b) the embedding of the HIGH-D-TUCKER instance into a Möbius-type simplex domain, in which two facets have been "identified" with each other - one can envision a high-dimensional Möbius strip with an instance of HIGH-D-TUCKER in its center, embedding in a high-dimensional simplex. A key step in the reduction is the extension of the labeling of HIGH-D-TUCKER to the remainder of the domain, in a way that does not introduce any artificial solutions, and such that solutions to HIGH-D-TUCKER can be traced back from solutions on other points on the domain. For this purpose, the authors of [Filos-Ratsikas and Goldberg, 2019] develop a rather complicated coordinate transformation, applied to the inputs read from the positions of the cuts. They establish how to compute the transformation and its inverse in polynomial time and how distances in the two coordinate systems (before and after the transformation) are polynomially related. In contrast, our reduction works with the rather clean domain of HIGH-D-TUCKER, avoiding all the unnecessary technical clutter of the domain used in [Filos-Ratsikas and Goldberg, 2019].

Simpler gadgetry. Another complication of the proof in [Filos-Ratsikas and Goldberg, 2019] is the use of blanket-sensor agents, which constrain the positions of the cuts in the coordinate-encoding region, to ensure that solutions to ε-CONSENSUS-HALVING do not encode points that lie too far from a specific region in the "middle" of the domain, called the "significant region"; this is achieved via appropriate feedback provided by these agents to the coordinate-encoding agents. To make sure that the blanket-sensor agents do not "cancel" each other, extra care must be taken on how the feedback of these agents is designed, giving rise to a series of technical lemmas. Our reduction does not need to use any such agents and is therefore significantly simpler in that regard as well.

Label sequence robustness. The reduction in [Filos-Ratsikas and Goldberg, 2019] requires knowledge of the label sequence, i.e., whether the first cut that occurs in the c-e region has the label $+$ or $-$ on its left side. This is fundamental for the design of the gates, as they read the inputs as the distances from the left endpoints of the corresponding designated intervals, unlike our interpretation, which measures the difference between the value of the two labels. Thus, for the disorientation of the domain and to deal with sign flips that happen due to the stray cuts, the authors of [Filos-Ratsikas and Goldberg, 2019] employ a pre-processing circuit that uses the first coordinate-detecting agent as a reference agent, when performing computations. This is again not needed in our case; our equivariant gates ensure that even when the corresponding point lies on the boundary of the HIGH-D-TUCKER domain, the output is computed correctly in a much simpler way.

3.3 Arithmetic Gates

In this section we show how to construct gates which perform various operations on numbers in $[-1,1]$ with error at most $g(\varepsilon) = 16\varepsilon$, where ε is the error we allow in a CONSENSUS-HALVING solution. Some of these gates will be immune to "corruption" by a stray cut, while others might get corrupted and not work properly.

Recall that in any solution S, any unit length interval I of the domain represents value $v(I) \in [-1,1]$. We will now show how to perform computations with these values. We let $T : \mathbb{R} \to [-1,1]$, $z \mapsto \max(-1, \min(1,z))$, i.e., $T[z]$ is the *truncation* of $z \in \mathbb{R}$ in $[-1,1]$. We will also abuse notation and use $T[x] = (T[x_1], \ldots, T[x_N])$ for $x \in [-1,1]^N$. At this stage, we assume that ε is sufficiently small for the gates to work (namely $\varepsilon \leq 2^{-10}$ is enough).

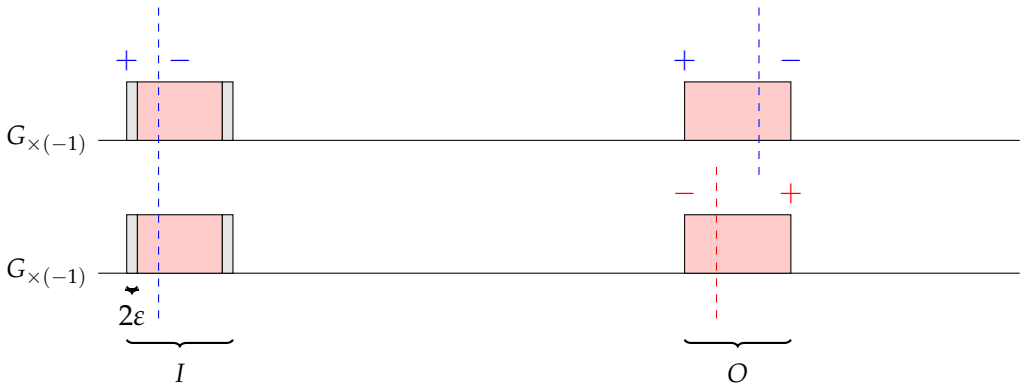

Figure 2: A ***Multiplication by*** -1 $[G_{\times(-1)}]$ gate. The gray shaded regions are not part of the agents valuation on I, but are shown for clarity. The input to the gate is $-1/2$ and the output is $1/2 \pm 4\varepsilon$. Two different sets of cut are shown (top and bottom), to emphasize the fact that the gadgets are resilient to flips in the parity of the cut sequence. On both cases, the parity sequence on the left is $+/-$; on the top, the parity sequence on the right is also $+/-$, and the cut is placed in the rightmost half in O, whereas on the bottom, the parity sequence on the right is $-/+$, and the cut is placed in the leftmost half in O.

We will design *basic gates*, namely ***Multiplication by*** -1 $[G_{\times(-1)}]$, ***Constant*** $\zeta \in [-1,1] \cap \mathbb{Q}$ $[G_\zeta]$ and ***Addition*** $[G_+]$, and *additional gates*, namely ***Copy*** $[G_{\mathbf{copy}}]$, ***Multiplication by*** $k \in \mathbb{N}$ $[G_{\times k}]$ and Boolean Gates, ***Negation*** $[G_\neg]$, ***AND*** $[G_\wedge]$ and ***OR*** $[G_\vee]$.

3.3.1 Basic Gates

δ-Volume Gate $[G_\delta]$: Let $\delta \in [2\varepsilon, 1]$. Let I and O be disjoint intervals of length 1. The agent for this gate has a block of length $1-\delta$ and height $\frac{1}{2-\delta}$ centered in interval I and a block of length 1 and (same) height $\frac{1}{2-\delta}$ in interval O.

It is easy to check that since $\delta \geq 2\varepsilon$, at least one cut must lie strictly within O in any solution. Furthermore, this gate has the notable property that it cannot be corrupted. From this construction, we obtain the following gates:

- ***Multiplication by*** -1 $[G_{\times(-1)}]$**:** Set $\delta = 2\varepsilon$. Then, in any solution it holds that $v(O) = T[-v(I) \pm 4\varepsilon]$. See Fig. 3 for an illustration.

- ***Constant*** $\zeta \in [-1,1] \cap \mathbb{Q}$ $[G_\zeta]$**:** To create a constant, we use an input interval I in the Constant-Creation region. Let j be such that this gate is part of circuit-simulator C_j. In this case, we use input $I := [N + (j-1), N + j]$ (the region corresponding to const_j). If $v(I) = +1$ (i.e., $\text{const}_j = +1$), then this gate will work properly and cannot be corrupted.

  - For $\zeta \leq 0$, we let $\delta = \max(1+\zeta, 2\varepsilon)$ and obtain that $v(O) = T[\zeta \pm 4\varepsilon]$.
  - For $\zeta > 0$, use $G_{-\zeta}$ and then $G_{\times(-1)}$, for a total error of at most 8ε. Note that if $|v(I)| = 1$, then this gate can also be used to obtain $T[v(I) \times \zeta \pm 8\varepsilon]$.

  See Fig. 2 for an illustration.

Figure 3: A ***Constant*** $\zeta \in [-1,1] \cap \mathbb{Q}$ $[G_\zeta]$ gate. The gray shaded regions are not part of the agents valuation on I, but are shown for clarity. I is part of the Constant-Creation region and therefore (in a well-behaved case), it is not intersected by any cuts. The value block of the agent on the left is labeled entirely by "+", and the cut on the right side assumes the corresponding position in favor of "−"m to balance out the discrepancy. In the example, $\zeta = -3/4$ and therefore $\delta = 1/4$, and the output on the right is $-3/4$.

Addition $[G_+]$**:** Let I_1 and I_2 be the two length-1 intervals encoding the two inputs. Let I' be an interval of length 2 that is disjoint from I_1 and I_2. Let J be an interval of length 3 that is disjoint from I_1, I_2 and I'. We first use a $G_{\times(-1)}$-gate with input I_1 and output $I'[0,1]$, and another one with input I_2 and output $I'[1,2]$.

To compute addition we create a new agent with valuation function that has height $1/5$ in I' and in J, and height 0 everywhere else. Note that since $\varepsilon < 1/5$, in any solution there must be a cut in J. We say that the gate is corrupted, if there are at least two cuts lying strictly in the interval J. The output of the gate will be in interval $O = J[1,2]$. If the gate is not corrupted, then by construction we have $v(O) = T[v(I_1) + v(I_2) \pm 16\varepsilon]$. See Fig. 4 for an illustration.

3.3.2 Additional Gates

Using the gates presented above, we can also implement the following operations.

Copy $[G_{\mathbf{copy}}]$**:** We can copy a value by using two successive $G_{\times(-1)}$-gates. The error will be at most 8ε. This gate can not be corrupted.

Multiplication by $k \in \mathbb{N}$ $[G_{\times k}]$**:** This can be implemented by a chain of $k-1$ additions that repeatedly add the input to the intermediate summation. It is easy to check that the error is at most $(k-1)16\varepsilon$.

Boolean Gates: For the Boolean gates, the bits $\{0,1\}$ will be represented by the values -1 (for 0) and $+1$ (for 1). We say that an input value $v(I) \in [-1,1]$ is a *perfect bit*, if $v(I) \in \{-1,1\}$. The Boolean gates we will construct have the following very nice property: if all inputs are perfect bits, then the output is a perfect bit (and is the correct output).

- ***Negation Gate*** $[G_\neg]$**:**. This can be done by first using a $G_{\times(-1)}$-gate and then a $G_{\times 2}$-gate. The error will be at most 20ε, so this will work as intended as long as $20\varepsilon \leq 1$.
- ***AND Gate:*** $[G_\wedge]$**:** Let b_1 and b_2 be the two inputs. We perform the computation $((b_1 + b_2) - 1/2) \times 4$ by using two G_+-gates, a $G_{-1/2}$-gate and a $G_{\times 4}$-gate. The error will be at most $12 \times 16\varepsilon$, so the gate will work as intended as long as $12 \times 16\varepsilon \leq 1$.

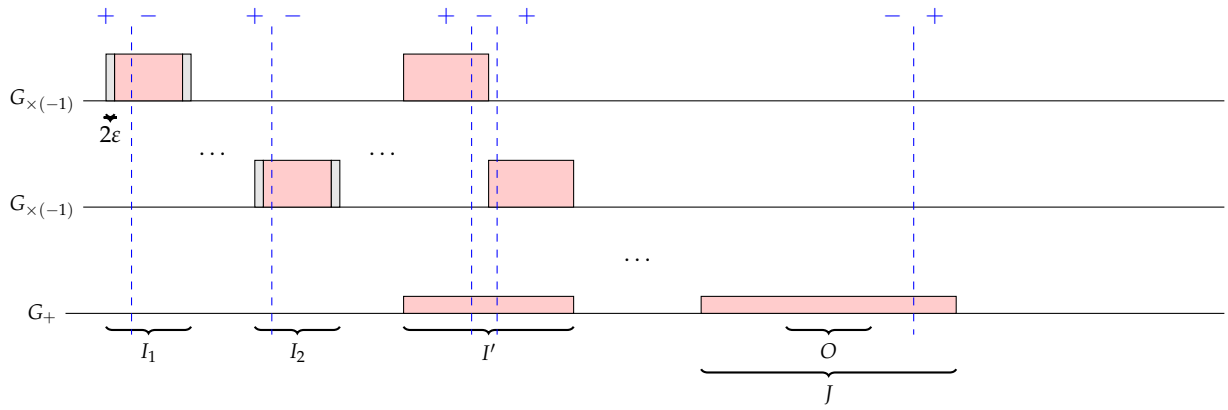

Figure 4: A ***Addition*** $[G_+]$ gate. An arbitrary sequence of labels is shown. First, we use two $G_{\times(-1)}$ gates for the two inputs and the outputs of those gates are "read together" in a single interval I'. The output of the G_+ gate is read from O. In the example, the first input is $-1/2$ and the second input is $-3/4$. The cut in J is placed appropriately to balance out the excess of "$+$" in I', but the output of the gate is actually -1, i.e., the gate applies truncation to the output of the addition as it is smaller than -1.

- ***OR Gate*** $[G_\vee]$***:*** This can easily be obtained by using $G_\wedge$ and $G_\neg$.

Note that all the arithmetic gates we have presented have error at most $g(\varepsilon) = 16\varepsilon$, with the exception of $G_{\times k}$, which has an error of $(k-1)g(\varepsilon)$. Thus, we have to be careful whenever we use this gate for some non-constant k.

Remark 1 (Equivariant Gates). Note that the operation performed by any gate is *equivariant*. Namely, if we flip the sign of all inputs, the same output is still valid, but with a flipped sign. For $G_{\times -1}$ and G_+ gates this is obvious. For G_ζ, we have to recall that const_j is the input to the gate. With this interpretation, the equivariance is once again obvious. For the $G_\wedge$-gate the equivariance is a bit more subtle. Note that it uses a $G_{-1/2}$-gate, which uses const_j. Thus, in this case again, if we flip the sign of the inputs b_1, b_2 and const_j, the sign of the output is flipped too.

This property of the gates is not a coincidence. It follows from the way we are encoding values. Note that in any solution S, if we swap the labels $+$ and $-$, the solution remains valid. The only thing that has changed is that for any gate, the sign of all inputs and outputs has been flipped.

It follows that any circuit that we construct out of these gates will be equivariant. Namely, the computation will still be valid if we flip the sign of all inputs (including const_j) and all outputs.

3.4 Circuit-Simulators

In this section we describe the functionality of the circuit-simulators and what it achieves. Recall that every circuit-simulator C_j reads-in inputs $x_1, \ldots, x_N \in [-1, 1]$ from the Coordinate-Encoding region and $\text{const}_j \in [-1, 1]$ from the Constant-Creation region. The circuit-simulator then performs some computations and outputs $[C_j(x, \text{const}_j)]_1, \ldots, [C_j(x, \text{const}_j)]_N \in [-1, 1]$ into the Feedback region. If the circuit-simulator C_j is corrupted (i.e., one of the stray cuts interferes with it), then we will not claim anything about the outputs of C_j. In that case, we will only use the fact that all the outputs must lie in $[-1, 1]$ (which is guaranteed by the way values are represented).

If the circuit-simulator C_j is not corrupted, then we know that all gates will perform correct computations, and we also know that $\text{const}_j \in \{-1, +1\}$. In our construction of C_j, we will be

assuming that $\text{const}_j = +1$. However, we will show later that even if $\text{const}_j = -1$, C_j will output something useful.

Phase 1: equi-angle displacement. In the first phase, C_j applies a small displacement to its input x. Namely, for every $i \in [N]$, C_j computes $\widehat{x}_i \approx T[x_i + j\alpha]$, where $\alpha = \frac{1}{16p(N)}$. This is achieved by using a $G_{j\alpha}$-gate to create the constant $j\alpha$ (by using const_j), followed by a G_+-gate to perform the addition $x_i + j\alpha$. However, since $\varepsilon > 0$, the gates might make some error in the computations. Nevertheless, by construction of the gates, we immediately obtain:

Claim 1. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *Then the equi-angle displacement phase outputs $\widehat{x}_i = T[x_i + j\alpha \pm 2g(\varepsilon)]$ for all i.*

Phase 2: bit extraction. In the second phase, C_j extracts the three most significant bits from each $\widehat{x}_i \in [-1, 1]$. These three bits tell us where $\widehat{x}_i$ lies in $[-1, 1]$, namely in which of the eight possible standard intervals of length $1/4$: $[-1, -3/4]$, $[-3/4, -1/2]$, $[-1/2, -1/4]$, $[-1/4, 0]$, $[0, 1/4]$, $[1/4, 1/2]$, $[1/2, 3/4]$, $[3/4, 1]$. Instead of the usual $\{0, 1\}$, our bits will take values in $\{-1, +1\}$. The first bit $b_1 \in \{-1, +1\}$ indicates whether $\widehat{x}_i$ is positive or negative. If $b_1 = +1$, then $\widehat{x}_i \in [0, 1]$. If $b_1 = -1$, then $\widehat{x}_i \in [-1, 0]$. The second bit $b_2 \in \{-1, +1\}$, then indicates in which half of that interval $\widehat{x}_i$ lies. Thus, if $b_1 = +1$ and $b_2 = -1$, then $\widehat{x}_i \in [0, 1/2]$. Note that some of the bits are not well-defined if $\widehat{x}_i \in B$ where $B = \{-3/4, -1/2, -1/4, 0, 1/4, 1/2, 3/4\}$. Thus, we cannot expect the bit extraction to succeed in this case. In fact, the bit extraction will fail if $\widehat{x}_i$ is sufficiently close to any point in B.

The bit extraction for $\widehat{x}_i$ is performed as follows.

1. $b_1 \approx T[\widehat{x}_i \times \lceil 1/g(\varepsilon) \rceil]$ (use $G_{\times \lceil 1/g(\varepsilon) \rceil}$-gate)
2. $\widehat{x}'_i \approx T[\widehat{x}_i - b_1/2]$ (use $G_{-1/2}$-gate and G_+-gate)
3. $b_2 \approx T[\widehat{x}'_i \times \lceil 1/g(\varepsilon) \rceil]$
4. $\widehat{x}''_i \approx T[\widehat{x}'_i - b_2/4]$
5. $b_3 \approx T[\widehat{x}''_i \times \lceil 1/g(\varepsilon) \rceil]$

Note that to compute $-b_1/2$ and $-b_2/4$ we just use the corresponding constant gate, namely $G_{-1/2}$ and $G_{-1/4}$ (with input b_1 or b_2 respectively, instead of const_j). The computation may be incorrect if b_1 or b_2 are not in $\{-1, 1\}$, but in that case the bit-extraction has already failed anyway.

We can show that the bit-extraction succeeds if $\widehat{x}_i$ is sufficiently far away from any point in B. Letting $\text{dist}(t, B) = \min_{p \in B} |t - p|$, we obtain:

Claim 2. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *If* $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$, *then the bit-extraction phase for $\widehat{x}_i$ outputs the correct bits for $T[x_i + j\alpha]$.*

Proof. Since $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$, it follows by Claim 1 that $\text{dist}(\widehat{x}_i, B) \geq 6g(\varepsilon)$, and in particular $|\widehat{x}_i| \geq 6g(\varepsilon)$. In step 1 we use a $G_{\times \lceil 1/g(\varepsilon) \rceil}$-gate on input $\widehat{x}_i$. By construction of the gate, it follows that $b_1 = T[\widehat{x}_i \times \lceil 1/g(\varepsilon) \rceil \pm (\lceil 1/g(\varepsilon) \rceil - 1)g(\varepsilon)] = T[\widehat{x}_i \times \lceil 1/g(\varepsilon) \rceil \pm 1]$ where we used $|(\lceil 1/g(\varepsilon) \rceil - 1)g(\varepsilon)| \leq 1$. Since $|\widehat{x}_i| \geq 6g(\varepsilon)$, it holds that $|\widehat{x}_i \times \lceil 1/g(\varepsilon) \rceil| \geq 6 \geq 2$. Thus, $b_1 \in \{-1, +1\}$ is the correct bit for $\widehat{x}_i$.

In step 2 we compute $\widehat{x}'_i = T[\widehat{x}_i - b_1/2 \pm 2g(\varepsilon)]$. Thus, we have $\text{dist}(\widehat{x}'_i, B) \geq 4g(\varepsilon)$, and in particular $|\widehat{x}'_i| \geq 4g(\varepsilon)$, which implies that $b_2 = T[\widehat{x}'_i \times \lceil 1/g(\varepsilon) \rceil \pm 1] \in \{-1, +1\}$ is the correct first bit for $\widehat{x}'_i$ and the correct second bit for $\widehat{x}_i$.

In step 4 we compute $\widehat{x}_i'' = T[\widehat{x}_i' - b_2/2 \pm 2g(\varepsilon)]$. Thus, $|\widehat{x}_i''| \geq 2g(\varepsilon)$, which implies that $b_3 = T[\widehat{x}_i'' \times \lceil 1/g(\varepsilon) \rceil \pm 1] \in \{-1, +1\}$ is the correct first bit for $\widehat{x}_i''$, i.e., the correct second bit for $\widehat{x}_i'$ and the correct third bit for $\widehat{x}_i$.

Finally, note that if $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$, then $T[x_i + j\alpha]$ and $\widehat{x}_i$ must lie in the same standard interval of length $1/4$, i.e., they must have the same three bits. □

Phase 3: simulation of λ. Recall that $\lambda : [8]^N \to \{\pm 1, \ldots, \pm N\}$ is the Boolean circuit computing the HIGH-D-TUCKER labeling. We interpret $[8]$ as a subdivision of $[-1, 1]$ into standard intervals of length $1/4$. Namely, 1 corresponds to $[-1, -3/4]$, 2 to $[-3/4, -1/2]$, etc. Thus, $[8]^N$ can be interpreted as a subdivision of $[-1, 1]^N$ into hypercubes of side-length $1/4$. With this in mind, we define $\overline{\lambda} : ([-1, 1] \setminus B)^N \to \{\pm 1, \ldots, \pm N\}$, so that for any $x \in ([-1, 1] \setminus B)^N$, $\overline{\lambda}(x)$ is the label that λ assigns to the hypercube containing x.

We can assume that the inputs of λ consist of three bits each, such that the number represented by these three bits yields an element in $[8]$ (by using $[8] \equiv \{0, 1, \ldots, 7\}$). The three bits $b_1, b_2, b_3 \in \{-1, +1\}$ extracted from $T[x_i + j\alpha]$ tell us exactly in which interval $T[x_i + j\alpha]$ lies. Note that if we were to map those bits to $\{0, 1\}$ (where $-1 \mapsto 0$ and $+1 \mapsto 1$), then the bit-string $b_1 b_2 b_3$ would correspond to the number associated with the interval and would thus be the correct corresponding input to the circuit.

We re-interpret the circuit λ as working on bits $\{-1, +1\}$, where -1 corresponds to 0. Clearly, we can implement this circuit in C_j with our Boolean gates. As long as the inputs to every gate are perfect bits (i.e., in $\{-1, +1\}$), the output of the gate will also be a perfect bit, and will correspond to the result of the operation computed by the gate. The inputs to the circuit will be exactly the bits obtained in the bit extraction phase for each $\widehat{x}_i$. Thus, it follows that if the bit extraction phase succeeds, then the simulation of λ will always have a correct output. In other words, using Claim 2, we obtain:

Claim 3. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *If* $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$ *for all $i \in [N]$, then the simulation phase outputs $\overline{\lambda}(T[x + j\alpha])$.*

Phase 4: output into the Feedback region. For convenience, we will assume that the output of the Boolean circuit λ is encoded in a particular way. It is easy to see that this is without loss of generality, since we can always modify λ so that it follows this encoding. The output of λ is an element in $\{\pm 1, \ldots, \pm N\}$. The encoding we choose uses $2N$ bits $y_1^a, y_1^b, y_2^a, y_2^b, \ldots, y_N^a, y_N^b$ to encode such an element. The element $+i$ is represented by $y_i^a = y_i^b = 1$ and $y_\ell^a = 1$, $y_\ell^b = 0$ for all $\ell \neq i$. The element $-i$ is represented by $y_i^a = y_i^b = 0$ and $y_\ell^a = 0$, $y_\ell^b = 1$ for all $\ell \neq i$.

Recall that in the simulation of λ inside C_j, we actually use the bits $\{-1, +1\}$ instead of $\{0, 1\}$. Thus, output $+i$ is represented by $y_i^a = y_i^b = +1$ and $y_\ell^a = +1$, $y_\ell^b = -1$ for all $\ell \neq i$. Whereas the element $-i$ is represented by $y_i^a = y_i^b = -1$ and $y_\ell^a = -1$, $y_\ell^b = +1$ for all $\ell \neq i$. For any $i \in [N]$ and any $z \in ([-1, 1] \setminus B)^N$ define $\overline{\lambda}_i(z)$ to be:

- $\overline{\lambda}_i(z) = +1$ if $\overline{\lambda}(z) = +i$
- $\overline{\lambda}_i(z) = -1$ if $\overline{\lambda}(z) = -i$
- $\overline{\lambda}_i(z) = 0$ otherwise.

Then, by Claim 3 we obtain that $T[y_i^a + y_i^b] = \overline{\lambda}_i(T[x + j\alpha])$.

In this last phase, for each $i \in [N]$ we compute $T[y_i^a + y_i^b]$ and copy this value into the Feedback region, namely into interval $F_i(j)$ (recall that C_j is the current circuit-simulator). For this we first use a G_+-gate and then a G_{copy}-gate. Thus, we immediately obtain:

Claim 4. *Assume that the circuit-simulator C_j is not corrupted and* $\text{const}_j = +1$. *If* $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$ *for all* $i \in [N]$, *then* $[C_j(x, \text{const}_j)]_i := v(F_i(j)) = T[\overline{\lambda}_i(T[x + j\alpha]) \pm 2g(\varepsilon)]$ *for all* $i \in [N]$.

3.5 Proof of Correctness

In this section we prove that the reduction works, i.e., from any solution to the Consensus-Halving instance, we can obtain a solution to the original high-D-Tucker instance in polynomial time. In order to do this, we consider two cases and show that we can retrieve a solution in both cases. The first case corresponds to a "well-behaved" solution where there are no stray cuts. The second case corresponds to a solution with stray cuts.

We set $p(n) = 4n^2$ and pick ε such that $16g(\varepsilon) \leq \frac{1}{16p(n)}$, i.e., $\varepsilon \leq \frac{1}{2^{14}n^2}$.

Lemma 4. *Let S be any ε-approximate solution for the* Consensus-Halving *instance. If S does not have any stray cuts, then it yields a solution to the* high-D-Tucker *instance in polynomial time.*

Proof. Since there are no stray cuts, none of the circuit-simulators is corrupted. In particular, there is no cut strictly inside the Constant-Creation region. Thus, without loss of generality we can assume that the whole Constant-Creation region is labeled $+$. This means that all circuit-simulators read-in the constant $+1$, i.e., $\text{const}_j = +1$ for all j.

Since the circuit-simulators are not corrupted, the only way in which they can fail is if the bit extraction phase fails. Let $x \in [-1, 1]^n$ be the point represented by the Coordinate-Encoding region in S. Let $z^{(j)} = T[x + j\alpha]$.

Consider any $i \in [n]$. First of all, note that if $z_i^{(j)} \in \{-1, +1\}$, then the bit extraction will not fail (for dimension i). So, we ignore any such points. For all $j \neq \ell$ such that $z_i^{(j)}, z_i^{(\ell)} \notin \{-1, +1\}$ it holds that

$$\alpha \leq |z_i^{(j)} - z_i^{(\ell)}| \leq p(n)\alpha.$$

Since $p(n)\alpha \leq 1/16$ and $\alpha = \frac{1}{16p(n)} \geq 16g(\varepsilon)$, it follows that there exists at most one j^* such that $z_i^{(j^*)}$ lies too close to $B = \{-3/4, -1/2, -1/4, 0, 1/4, 1/2, 3/4\}$. Namely, there exists $j^* \in [p(n)]$ such that for all $j \in [p(n)] \setminus \{j^*\}$, we have $\text{dist}(z_i^{(j)}, B) \geq 8g(\varepsilon)$. By Claim 2 it follows that the bit extraction phase in dimension i fails in at most one circuit-simulator.

We thus obtain that the bit extraction (in any dimension) fails in at most n circuit-simulators. Let $X \subseteq [p(n)]$ be such that the bit extraction does not fail in C_j for all $j \in X$. Then, we have shown that $|X| \geq p(n) - n \geq p(n) - 2n$. By Claim 3, we get that for all $j \in X$, phase 3 of the circuit-simulator C_j outputs $\overline{\lambda}(z^{(j)})$, i.e., the correct label for $z^{(j)}$ (which lies in $([-1, 1] \setminus B)^n$).

Since $\|z^{(j)} - z^{(\ell)}\|_\infty \leq p(n)\alpha \leq 1/16$ for all $j, \ell \in [p(n)]$, all the points $z^{(j)}$ lie in (pairwise) adjacent hypercubes of $[-1, 1]^n$ (as defined in phase 3). Thus, in order to find a solution to the nD-Tucker instance, it suffices to find $j, \ell \in X$ such that $\overline{\lambda}(z^{(j)}) = -\overline{\lambda}(z^{(\ell)})$.

By Claim 4, we have that for every $j \in X$ the circuit-simulator C_j outputs $[C_j(x, \text{const}_j)]_1$, $\ldots$, $[C_j(x, \text{const}_j)]_n$ such that for all $i \in [n]$ we have $[C_j(x, \text{const}_j)]_i \approx \overline{\lambda}_i(z^{(j)})$ (up to error $2g(\varepsilon)$). It holds that for every $j \in X$, there exists $i_j \in [n]$ such that $|\overline{\lambda}_{i_j}(z^{(j)})| = 1$. Thus, there exist $i \in [n]$ and $X_i \subseteq X$ such that $|X_i| \geq |X|/n$, $|\overline{\lambda}_i(z^{(j)})| = 1$ for all $j \in X_i$ and $|\overline{\lambda}_i(z^{(j)})| = 0$ for all $j \in X \setminus X_i$. If there exist $j, \ell \in X_i$ such that $\overline{\lambda}_i(z^{(j)}) = -\overline{\lambda}_i(z^{(\ell)})$, then $z^{(j)}$ and $z^{(\ell)}$ have opposite labels and we are done. Thus, assume that $\overline{\lambda}_i(z^{(j)}) = +1$ for all $j \in X_i$ (the case with -1 instead works analogously). It follows that $[C_j(x, \text{const}_j)]_i \geq 1 - 2g(\varepsilon)$ for all $j \in X_i$. Since we also have

$[C_j(x, \text{const}_j)]_i \geq -2g(\varepsilon)$ for all $j \in X \setminus X_i$, we can write:

$$\begin{aligned}\sum_{j=1}^{p(n)} [C_j(x, \text{const}_j)]_i &= \sum_{j \in X} [C_j(x, \text{const}_j)]_i + \sum_{j \in [p(n)] \setminus X} [C_j(x, \text{const}_j)]_i \\ &\geq |X_i| - p(n)2g(\varepsilon) - |[p(n)] \setminus X| \\ &\geq \frac{p(n) - 2n}{n} - p(n)2g(\varepsilon) - 2n \geq 2n - 2 - 2^7/2^{14} > p(n)\varepsilon\end{aligned}$$

for n sufficiently large, where we used $p(n) = 4n^2$ and $\varepsilon \leq \frac{1}{2^{14}n^2}$. However, recall that feedback agent f_i ensures that $\sum_{j=1}^{p(n)} [C_j(x, \text{const}_j)]_i = \sum_{j=1}^{p(n)} v(F_i(j)) \in [-p(n)\varepsilon, p(n)\varepsilon]$. So, we have obtained a contradiction. □

Lemma 5. *Let S be any ε-approximate solution for the* Consensus-Halving *instance. If S has at least one stray cut, then it yields a solution to the* high-D-Tucker *instance in polynomial time.*

Proof. As explained earlier, stray cuts can affect the circuit-simulators in two ways. First of all, a stray cut can corrupt a circuit-simulator. Since there are at most n stray cuts, at most n circuit-simulators can be corrupted. The other way in which the circuit-simulators can be affected, is if there are stray cuts in the Constant-Creation region and $\text{const}_j = -1$ for some C_j, and $\text{const}_\ell = +1$ for some other C_ℓ.

Let us now take a look at what happens if a circuit-simulator C_j has $\text{const}_j = -1$. Since all the gates in C_j are equivariant (Remark 1), it follows that C_j with inputs $x_1, \ldots, x_n$ and const_j is also equivariant. This implies that $[C_j(x, -1)]_i = -[C_j(-x, +1)]_i$ for all $i \in [n]$. Thus, the output of C_j is the negation of a valid output that C_j would have had if its inputs were $-x$ and $\text{const}_j = +1$ instead. We use the term "a valid output" instead of "the output", because C_j can have multiple valid outputs for a single input (because every gate is allowed a small error). So, we are saying that the output of C_j on inputs $x, -1$ is the negation of a valid output on inputs $-x, +1$. Thus, we immediately obtain that the analogous statements for Claims 1,2,3 and 4 also hold in this case. In other words, we get that if circuit-simulator C_j is not corrupted, and if $\text{dist}(T[x_i + j\alpha], B) \geq 8g(\varepsilon)$, then $[C_j(x, +1)]_i = \overline{\lambda}_i(T[x + j\alpha]) \pm 2g(\varepsilon)$ and $[C_j(x, -1)]_i = -\overline{\lambda}_i(T[-x + j\alpha]) \pm 2g(\varepsilon)$.

Consider any $i \in [n]$. Using the same argument as in the proof of Lemma 4, it is easy to show that at most one of the points $T[x_i + \alpha], \ldots, T[x_i + p(n)\alpha]$ and $T[x_i - \alpha], \ldots, T[x_i - p(n)\alpha]$ lies within distance at most $8g(\varepsilon)$ of B. It follows that at most one of the points $T[x_i + \alpha], \ldots, T[x_i + p(n)\alpha]$ and $T[-x_i + \alpha], \ldots, T[-x_i + p(n)\alpha]$ lies within distance at most $8g(\varepsilon)$ of B. Thus, we again have that the bit extraction phase fails in at most n circuit-simulators.

Let $X \subseteq [p(n)]$ be such that for all $j \in X$, C_j is not corrupted and the bit extraction does not fail in C_j. Then we have shown that $|X| \geq p(n) - 2n$. The same argument as in the proof of Lemma 4 yields that there must exist $i \in [n]$ and $j, \ell \in X$ such that $[C_j(x, \text{const}_j)]_i = +1 \pm 2g(\varepsilon)$ and $[C_\ell(x, \text{const}_\ell)]_i = -1 \pm 2g(\varepsilon)$. If $\text{const}_j = \text{const}_\ell = +1$, then it follows that $\overline{\lambda}_i(T[x + j\alpha]) = +1$ and $\overline{\lambda}_i(T[x + \ell\alpha]) = -1$. Thus, we have obtained two points in adjacent hypercubes that have opposite labels. If $\text{const}_j = \text{const}_\ell = -1$, then it follows that $\overline{\lambda}_i(T[-x + j\alpha]) = -1$ and $\overline{\lambda}_i(T[-x + \ell\alpha]) = +1$. Thus, we again have obtained two points in adjacent hypercubes that have opposite labels.

Let us now see what happens in the case where $\text{const}_j = -\text{const}_\ell$. Without loss of generality assume that $\text{const}_j = +1$ and $\text{const}_\ell = -1$. Then, we obtain that $\overline{\lambda}(T[x + j\alpha]) = \overline{\lambda}(T[-x + \ell\alpha])$. Since there is at least one stray cut in the solution S, by Observation 1 we know that the point $x \in [-1, 1]^n$ encoded by the Coordinate-Encoding region must lie on the boundary of $[-1, 1]^n$. Thus, since $p(n)\alpha \leq 1/16$, $u = T[x + j\alpha]$ and $v = T[-x + \ell\alpha]$ must lie in hypercubes on the

boundary. Since $2p(n)\alpha \leq 1/8$, we know that u and $-v$ lie in adjacent hypercubes. However, by the boundary conditions of the nD-Tucker instance λ, we have $\overline{\lambda}(u) = \overline{\lambda}(v) = -\overline{\lambda}(-v)$. Thus, u and $-v$ yield adjacent hypercubes with opposite labels, and thus a solution to the nD-Tucker instance. Note that u and $-v$ cannot lie in the same hypercube, because of the boundary conditions of λ. □

4 Algorithms for Single-Block Valuations

In the previous section, we proved that even when we have 2-block Uniform valuations, the problem remains PPA-hard (even when we are allowed to use $n + n^{1-\delta}$ cuts, for some constant $\delta > 0$). In this section, we will consider the natural case that is not covered by our hardness, that of single-block valuations. Our main results of the section are (a) an algorithm for solving the problem to any precision ε (where ε appears polynomially in the running time), which is parameterized by the maximum number of intersection between the blocks of different agents and (b) a polynomial-time algorithm for 1/2-Consensus-Halving. The latter algorithm generalizes to the case of d-block uniform valuations (in fact, even to the case of piecewise constant valuations with d blocks) if one is allowed to use $d \cdot n$ cuts instead of n. Towards the end of the section, we also present a simple idea based on linear programming, which allows us to solve the Consensus-Halving problem in polynomial time, when we are allowed to use $2n - \ell$ cuts, for any ℓ which is constant.

4.1 A Parameterized Algorithm for ε-Consensus-Halving

We start with the algorithm for solving ε-Consensus-Halving that is parameterized by the *maximum intersection* between the probability measures. In particular, if d is the maximum number of measures with positive density at any point $x \in [0,1]$ and the maximum value of the densities is at most M then we provide a dynamic programming algorithm that computes an ε-approximate solution in time $O\left(\left(\frac{M}{\varepsilon}\right)^d \cdot \text{poly}(M/\varepsilon, n, d)\right)$. We start with the formal definition of the maximum intersection quantity. In this section we denote by f_i the probability density function of the probability measure μ_i.

Definition 7 (Maximum Intersection & Maximum Value)**.** We say that a single-block instance of ε-Consensus-Halving has *maximum intersection* d if for every $x \in [0,1]$ it holds that the set $\mathcal{R}(x) = \{i \in [n] \mid f_i(x) > 0\}$, has cardinality $|\mathcal{R}(x)| \leq d$. We also say that the instance has *maximum value* M if for every $i \in [n]$ and every $x \in [0,1]$ it holds that $f_i(x) \leq M$. This is equivalent to $b_i - a_i \geq 1/M$.

For the rest of the section, we often refer to the following quantity.

Definition 8 (Value of Balance)**.** For any ε-Consensus-Halving instance $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$, any vector of cuts $\vec{s}$ and any $z \in [0,1]$ let $b_i(\vec{s}; z)$ be the mass with respect to μ_i of the part of the interval $[z,1]$ labeled "+" minus the part of the interval labeled "−", when split with the cuts $\vec{s}$. Formally, $b_i(\vec{s}; z) = \mu_i([z,1]^+) - \mu_i([z,1]^-)$, given the set of cuts $\vec{s}$.

The first step of the algorithm is to discretize the interval $[0,1]$ into to intervals of length $\varepsilon/(2M)$. Hence, we split the interval $[0,1]$ in $m = 2M/\varepsilon$ equal subintervals of the form $p_\ell = [(\ell-1)/m, \ell/m]$, where $\ell \in [m]$. The following claim shows the sufficiency of our discretization and follows very easily from the above definitions.

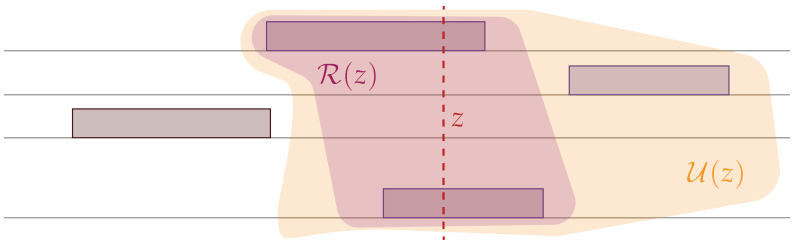

Figure 5: The definition of the sets $\mathcal{U}(z)$ and $\mathcal{R}(z)$.

Claim 5. *Let* $Q_m = \left\{\frac{\ell-1}{m} \mid \ell \in [m]\right\}$*, where* $m = M/\varepsilon'$ *and let* $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$ *be a single-block* ε-Consensus-Halving *instance with maximum value* M*. Then we define the* rounded *single-block instance* $\mathcal{C}' = \{\mu'_1, \ldots, \mu'_n\}$ *where* a'_i *and* b'_i *are equal to the number of* Q_m *that is closer to* a_i *and* b_i *respectively. Then every* ε-Consensus-Halving *solution of* $\mathcal{C}'$ *is an* $(\varepsilon + \varepsilon')$-Consensus-Halving *solution of* $\mathcal{C}$*. Additionally, there exists a solution to the* ε'-Consensus-Halving *problem where the positions of the cuts lie in the set* Q_m*, where* $m = M/\varepsilon'$*.*

Because of Claim 5 we will assume for the rest of this section that we are working with $\mathcal{C}'$ and we will focus on finding $(\varepsilon' = \varepsilon/2)$-Consensus-Halving solution with cuts in Q_m, where $m = 2M/\varepsilon$. This will give us an ε-approximate solution for the single-block instance $\mathcal{C}$. We also need the following definitions:

- ▷ for any $z \in [0,1]$ and any instance $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$ we define the set of measures that have positive mass in $[z,1]$ as follows: $\mathcal{U}(z;\mathcal{C}) = \{i \in [n] \mid \mu_i \in \mathcal{C} \wedge \mu_i([z,1]) > 0\}$ where we might drop $\mathcal{C}$ when it is clear from the context, see also Figure 5,
- ▷ for any $z \in [0,1]$ and any instance $\mathcal{C} = \{\mu_1, \ldots, \mu_n\}$ we define the set of measures that have positive density at z as follows: $\mathcal{R}(z;\mathcal{C}) = \{i \in [n] \mid \mu_i \in \mathcal{C} \wedge f_i(z) > 0\}$ where we might drop $\mathcal{C}$ when it is clear from the context, see also Figure 5.

We also define $\bar{Q}_m = \{z \mid z \in Q_m \vee -z \in Q_m\}$. Now we are ready to define our main recursive relation for solving the instance $\mathcal{C}'$. For this we define the function $\alpha(q_1, \ldots, q_d, z, t)$ where $q_j \in \bar{Q}_m$, $z \in Q_m$, and $t \in [n]$ and $\alpha : \bar{Q}_m^d \times Q_m \times [n] \to \{0,1\}$. The intuitive explanation of the value of α is the following:

> *$\alpha(q_1, \ldots, q_d, z, t)$ denotes whether it is possible to find t cuts $\vec{s}$ to split the interval $[z,1]$ such that: (1) if i is the jth element of $\mathcal{R}(z)$ it holds that $|b_i(\vec{s};z) - q_j| \leq \varepsilon$, (2) for every $i \in \mathcal{U}(z) \setminus \mathcal{R}(z)$ it holds that $|b_i(\vec{s};0)| = |b_i(\vec{s};z)| \leq \varepsilon$.*

The value of $\alpha(q_1, \ldots, q_d, z, t)$ can be recursively computed via the following procedure.

- ▶ for every $r \in Q_m$, with $r > z$ do
  - ▷ if there exists $i \in \mathcal{R}(z)$ and i is the jth element of $\mathcal{R}(z)$ but $i \notin \mathcal{R}(r)$ and by adding the cut r to the set of cuts then the condition $|b_i(r;z) - q_j| > \varepsilon$ holds then **continue**,
  - ▷ if there exists $i \in \mathcal{U}(z) \setminus \mathcal{R}(z)$ but $i \notin \mathcal{U}(r) \setminus \mathcal{R}(r)$ then **continue**
  - ▷ else for every $i \in \mathcal{R}(r)$, where i is the jth element of $\mathcal{R}(r)$
    - if $i \notin \mathcal{R}(z)$ then we set $q'_j = (-1)^t \mu_i([z,r])$
    - if i is the ℓth element of $\mathcal{R}(z)$ then we set $q'_j = (-1)^t \mu_i([z,r]) + q_\ell$

· call the function $\alpha(q'_1, \ldots, q'_d, r, t-1)$

- return **true** if at least one of the recursive calls is successful, and **false** otherwise.

This procedure evaluates the binary function α, but our goal is to solve the search problem that finds a set of cuts that form a solution to ε'-Consensus-Halving. This can be easily done by storing one possible solution whenever $\alpha =$ **true**. We call this possible solution $\beta(q_1, \ldots, q_d, z, t)$. More formally, the algorithm is shown in Algorithm 1.

ALGORITHM 1: Recursive Computation of α, β

Input: leftover balances $q_1, \ldots, q_d$, position of last cut z, leftover number of cuts t, required accuracy ε.
Output: value of $(\alpha(q_1, \ldots, q_d, z, t), \beta(q_1, \ldots, q_d, z, t))$ as described above.

```
if t = 0 then
    if q_j = 0 ∀j ∈ [d] then
        return (true, {})
    end
    return (false, {})
end
for every r ∈ Q_m with r > z do
    for j ∈ [d] do
        Let i be the jth element of R(z)
        if i ∉ R(r) and |b_i(r; z) − q_j| > ε then
            continue to next value of r
        end
    end
    for i ∈ U(z) \ R(z) do
        if i ∉ U(r) \ R(r) then
            return (false, {})
        end
    end
    for j ∈ [d] do
        Let i the jth element of R(r)
        if i ∉ R(z) then
            q'_j ← (−1)^t μ_i([z, r])
        end
        if i is the ℓth element of R(z) then
            q'_j ← (−1)^t μ_i([z, r]) + q_ℓ
        end
    end
    res ← (α(q'_1, ..., q'_d, r, t − 1), β(q'_1, ..., q'_d, r, t − 1))
    if res = true then
        return (true, β(q'_1, ..., q'_d, r, t − 1) ∪ {r})
    end
end
return (false, {})
```

From the above recursive algorithm we see that the evaluation order for the dynamic programming algorithm that computes $\alpha(q_1, \ldots, q_d, z, t)$ starts from $t = 0$ to $t = n$, $z = 1$ to $z = 0$ and $|q_j| = 1$ to $|q_j| = 0$.

Theorem 6. *There exists a dynamic programming algorithm that for any single-block instance $\mathcal{C}$ of ε-*Consensus-Halving *with maximum value M and maximum intersection d, computes a set of cuts that*

define a solution. The running time of the algorithm is $O\left(\left(\frac{2M}{\varepsilon}\right)^{d+2} n(n+d)\right)$.

4.2 A polynomial-time algorithm for 1/2-Consensus-Halving

In this subsection, we present a polynomial-time approximation algorithm for 1/2-Consensus-Halving, for the case of single-block valuations. We state the main theorem, which will be proven in the subsection.

Theorem 7. *There is a polynomial-time algorithm for* 1/2-Consensus-Halving*, when agents have single-block valuations.*

High-level description: The high-level idea of the algorithm is the following greedy strategy. We will consider the agents in order of non-decreasing height of their valuation blocks. Since each agent has a single block of value, this means that for two agents i and j that have their value block in intervals I_i and I_j with $i < j$, agent i will be considered before agent j. For each agent, we will attempt to "reserve" a large enough sub-interval of her value block (of total value at least $1/2$ for the agent) and split it in half (to two parts of equal value at least $1/4$ to the agent) using a cut, assigning each half to $+$ and $-$ respectively. At that point, this split will ensure that $|\mu_i(I_i^+) - \mu_i(I_i^-) \leq 1/2$, regardless of the labeling of the remaining part of the agent's value block I_i. A reserved sub-interval, or *reserved region* (RR) will never be intersected by any subsequent cut. This ensures that the guarantee $|\mu_i(I_i^+) - \mu_i(I_i^-) \leq 1/2$ for every agent i previously considered will continue to hold.

Reserving a large enough region for the first considered agent is straightforward. For any subsequent agent i, part of her value block interval I_i might already be "covered" by regions that were reserved in previous steps. Before inserting any new cut, we will *expand* some of the RRs which are contained in I_i (those that exhibit a different label on each of their endpoints), until we either ensure that the agent is approximately satisfied with the imbalance of labels that she sees, or these RRs cannot be expanded any longer. In the latter case, we will place the cut corresponding to agent i in I_i, on the midpoint of the "virtual" interval $U_i \subseteq I_i$, consisting of all the intervals of I_i not covered by RRs, "glued" together. Then, we will create a new RR, which will potentially also contain some of the RRs already present in I_i, which will be such that (a) the total value covered by RRs for agent i is $1/2$ and (b) the agent is approximately satisfied (up to a $1/2$) with the value she has for RRs in I_i.

4.2.1 The Algorithm

First, we will use the following terminology: A *reserved region* (RR) R is a designated interval in which no further cuts are allowed to be placed, other than those that already lie in the region. We will define the *parity* of R to be *odd* if the left-hand side of the leftmost cut intersecting R and the right-hand side of the rightmost cut intersecting R have different labels; otherwise, we will say that the parity of R is *even*.

We will let I_i denote the (single) interval in which agent i has positive value, and we will let $R_i = \bigcup_{R_j \text{ is an RR}}(R_j \cap I_i)$ be the part of I_i that is "covered" by RRs. We will also let $U_i = I_i \setminus R_i$ denote the *unreserved* part of I_i. Finally, we will say that an RR R is an *internal RR* of I_i, if it is entirely contained in I_i (including the case where the endpoint of the RR is the endpoint of the interval); otherwise, we will say that R is a *boundary RR*. Note that since RRs can be contained in other RRs, it is possible that an internal RR is contained in a boundary RR (see Fig. 6). An

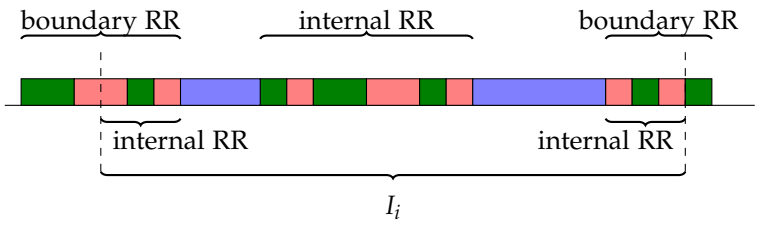

Figure 6: Internal and external reserved areas. The areas colored green have been assigned to one of the labels (e.g., "+") and the areas colored red had been assigned to the other (e.g., "-"). The areas of the interval I_i which are not yet assigned to any label are colored blue. The parity of the RRs can also be seen: the boundary RRs and the internal RR in the middle of the interval have odd parity, whereas the other two internal RRs (which are sub-regions of the boundary RRs) have even parity.

overview of the algorithm is given in Algorithm 2.

The algorithm considers the agents in non-increasing order of the heights of their value blocks. We will say that an agent i is $1/2$-*satisfied at step* j, if, after considering agent j, it holds that $|\mu_i(I^+) - \mu_i(I^-) \leq 1/2$, i.e., at least half of the value of the agent is balanced between $+$ and $-$. Let step i be the step when agent i is considered. At step i the algorithm does the following.

1. Expand internal RRs of odd parity, until agent i is $1/2$-satisfied, or until any internal RRs of odd parity can no longer be extended. See Section 4.2.2 below.

2. Place the cut in the midpoint of $\mathcal{U}_i$ (where $\mathcal{U}_i$ is obtained by considering U_i as a continuous interval, and disregarding R_i). See Section 4.2.3 below.

3. Create a new RR, by considering only U_i, possibly merging the intervals of the new RR with some RRs in R_i. See Section 4.2.4 below.

4.2.2 Expanding internal Reserved Regions of odd parity

Consider step i, when we are considering agent i, and her corresponding value block interval I_i. We will consider internal RRs of odd parity in I_i. For any such RR, we extend its endpoints at the same rate towards the left and the right, until we either:

(a) reach a point where the agent is $1/2$-satisfied or

(b) reach the endpoint of one or two other RRs, or,

(c) reach one of the endpoints of I_i.

If Condition (a) above is satisfied, then we do not consider any other RR and continue with the next agent. Otherwise, we continue with the next internal RR of odd parity, if any. Note that after this part of the algorithm, the cut corresponding to agent i has not been placed yet. See Fig. 7.

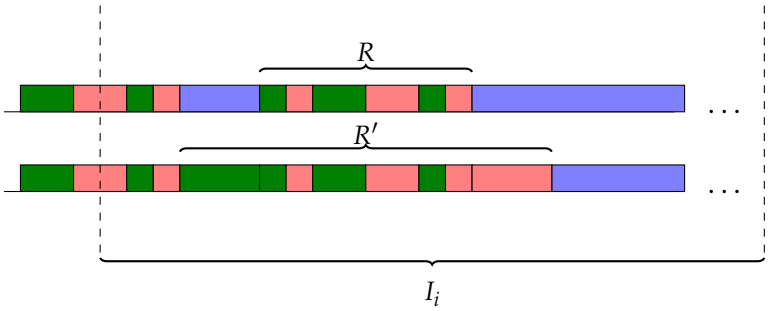

Figure 7: An expansion of an internal reserved region of odd parity. The areas colored green have been assigned to one of the labels (e.g., "+") and the areas colored red had been assigned to the other (e.g., "-"). The areas of the interval I_i which are not yet assigned to any label are colored blue. The reserved region R is expanded symmetrically on both sides, until it meets the left endpoint of the boundary region of I_i on the left, in which case the expansion stops (Condition (b) in Section 4.2.2). This results in the creation of new RRs on the left (a boundary RR and an internal RR contained in the boundary RR) that contain both the original RR R and its expansion R'. Crucially, the area of I_i that is covered by RRs has increased, while maintaining the same balance between the two labels.

4.2.3 Placing the cut

After we consider all internal RRs of odd parity, and if Condition (a) above has not been satisfied yet, we will use the agent's corresponding cut to balance out the remaining value block I_i, to ensure that the agent becomes $1/2$-satisfied. To do that, we consider agent i's valuation in I_i on U_i, and we place the cut on the midpoint y of U_i - one can envision this operation as considering U_i as a continuous interval (merging any sub-intervals separated by RRs in R_i) and splitting this interval in half via the placement of the cut. See Fig. 8.

4.2.4 Creating the new Reserved Region

After the cut has been placed, we need to reserve a corresponding region for agent i, to ensure that she is $1/2$-satisfied from the updated union of reserved regions R_i in I_i. To do that, we reserve an equal portion of U_i to the left and to the right of the position y of the cut, until the agent becomes $1/2$-satisfied. We argue in the proof of Lemma 10 that this is always possible. Intuitively, one can visualize that we extend the region from y to the left and to the right at the same rate, "jumping over" RRs, until we have reserved enough area to $1/2$-satisfy the agent. The effect of this process is a new RR, possibly containing previously reserved RRs and the newly reserved regions. See Fig. 8.

4.2.5 Correctness of the algorithm

Below, we argue about the correctness of the algorithm. We will prove the correctness via a series of lemmas.

Lemma 8. *After step i of the algorithm, the value of agent i for any internal RR in R_i is at most $1/2$.*

Proof. We will prove the lemma using induction. For $i = 1$, the statement is clearly true; there are no pre-existing RRs, so we place a cut in the midpoint y_1 of I_1 and reserve a region of total value

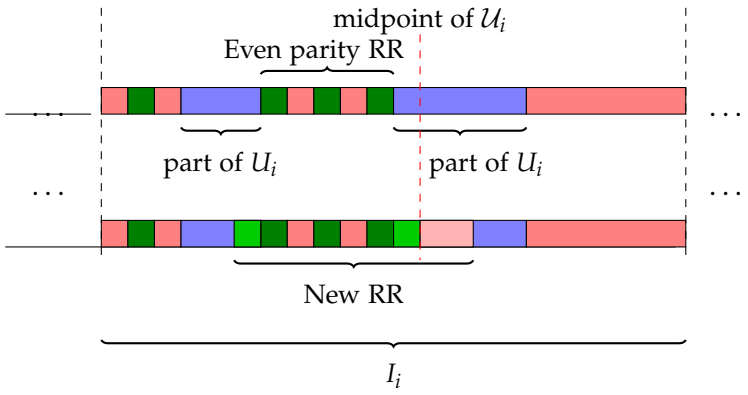

Figure 8: The placement of the cut in U_i and the creation of the new region. The parts of U_i are shown in blue and the midpoint of $\mathcal{U}_i$, where the cut is placed, is shown with a red dashed line. After the cut is placed, an equal amount of green (e.g., "+") and of red (e.g., "−") will be reserved to the left and the right of the position of the cut, "jumping over" RRs of even parity. The newly added parts of U_i (which are now part of R_i instead are shown in slightly different shades of their respective colors, for clarity. After the operation, the agent is 1/2-satisfied, which is guaranteed by her value in the union of restricted regions R_i.

1/4 for the agent to the left of y_1, and a region of total value 1/4 to the right of y_1, which together form the first RR R_1. This is the only internal RR in I_1 and hence the lemma follows.

Next, we will argue that the lemma holds for some $1 < j \leq n$. Consider agent j and its value interval I_j. Since we consider agents in non-increasing order of the height of their valuation blocks, from the induction hypothesis, it holds that the value of agent j for any RR which is internal for I_i, for any $i < j$, is at most 1/2 as well. Therefore, it is not possible for agent j to value any RR which is internal for I_j, at the beginning of step j, at more than 1/2. During step j, the algorithm might either expand existing internal RRs, create a new RR, or both, but in any case, by construction, the union of these RRs will never amount to more than 1/2 of agent j's value. □

Lemma 9. *Let R_b be any boundary RR in I_i at step i. Then, it holds that $|\mu_i(R_b^+) - \mu_i(R_b^-)| \leq 1/4$.*

Proof. First, note that since R_b is a boundary RR in I_i, it is part of some larger RR; let R be that RR, i.e., $R_b \subseteq R$. Also, note that R is "balanced", meaning that the total length of R labeled "+" and the total length of R labeled "−" are equal; this follows from the fact that in any of the steps (1) and (3) of the algorithm (Section 4.2.2 and Section 4.2.4) where the RRs are created or expanded, these operations are only done symmetrically with respect to the position of some cut. In other words, if R is internal for some agent j, it holds that $\mu_j(R^+) = \mu_j(R^-)$.

Assume by contradiction that $|\mu_i(R_b^+) - \mu_i(R_b^-)| > 1/4$ and assume without loss of generality that the excess is in favor of "+" over "−". This in particular means that more than 1/4 of the value of agent j in R_b is labeled "+", which means that R_b contains an interval of length larger than $1/4 \cdot v_i$ that is labeled "+", where v_j is the height of agent i's valuation block. Since R is balanced, this means that R must have length at least $1/2 \cdot v_i$. However, consider the step j when the R was last modified (either created or expanded) before step i and observe that R was an internal RR for I_j at the point. This means that agent j had value larger than $v_j \cdot 1/(2 \cdot v_i) \geq 1/2$ for R at the point, contradicting Lemma 8. □

Lemma 10. *After step (3) of the algorithm (Section 4.2.4), agent i is 1/2-satisfied.*

ALGORITHM 2: Polynomial-time algorithm for 1/2-CONSENSUS-HALVING

Input: The value blocks of the agents, as pairs (I_i, v_i), such that

- $I_i = [\alpha_i, b_i]$ is the interval.
- v_i is the height of the value block.

Output: The set of cuts $c_1, \ldots, c_n$, which satisfies that $|\mu_i(I^+) - \mu_i(I^-) \leq 1/2$ for every agent i.

Order the agents in non-increasing order of v_i and let wlog $\{1, \ldots, n\}$ be any such ordering.
for $i = 1$ *to* n **do**

```
/* Step (1), Expanding internal RRs of odd parity (see Section 4.2.2).   */
```
Let $\mathcal{R}^{\text{odd}}$ be the set of *internal* RRs of *odd parity* in I_i that can be expanded.
while $\mathcal{R}^{odd} \neq \emptyset$ **do**
Pick R in $\mathcal{R}^{\text{odd}}$
Expand R until it can no longer be expanded, i.e.,
- The agent becomes 1/2-satisfied.
- The expansion reaches the endpoint of some other RR.
- The expansion reaches the endpoint of I_i.

if *Agent i is 1/2-satisfied* **then**
Break and move on to the next agent.
end
Generate new RRs (if the expansion of R is merged with some other RR).
Update $\mathcal{R}^{\text{odd}}$.
end

```
/* Step (2), Placing the cut in U_i (see Section 4.2.3).                 */
```
Let $\mathcal{U}_i$ be the interval obtained by joining the intervals of U_i.
Let y be the midpoint of $\mathcal{U}_i$.
Place cut c_i on y.

```
/* Step (3), Creating the new RR (see Section 4.2.4).                    */
```
Consider some $x \in \mathbb{R}$ and the intervals $[y - x, y], [y, y + x] \subseteq \mathcal{U}_i$.
Pick x to satisfy $v_i \cdot (y - x) + v_i \cdot (y + x) + \mu_i(R_i) = 1/2$.
Let y_1 and y_2 be the points in U_i corresponding to $y - x$ and $y + x$ respectively.
Create a new RR $R' = [y_1, y_2]$ (possibly containing other RRs of even parity).
end

Proof. Consider the imbalance between the portions of I_i assigned to the two labels after step (1) of the algorithm, and assume without loss of generality that there is an excess of the label "+" over the label "−", i.e., $\mu_i(I_i^+) > \mu_i(I_i^-)$, as otherwise the agent is perfectly satisfied and we are done. Since by definition, there can be at most 2 external RRs for I_i, and since all internal RRs have a perfect balance of "+" and "−" for agent i, it follows from Lemma 9 that $\mu_i(R_i^+) - \mu_i(R_i^-) \leq 1/2$, i.e., the difference in value for the agent in the reserved regions of I_i is at most $1/2$. If $\mu_i(R_i) \geq 1/2$, then we are done, as the agent is 1/2-satisfied. Otherwise, we place the cut in step (2) of the algorithm on the midpoint y of $U_i = I_i \setminus R_i$ and we reserve an equal portion of I_i to the left and to the right of y, until we establish that $\mu_i(R_i) = 1/2$.

The only thing left to argue is that the creation of the RR R in step (3) of the algorithm that will make the agent 1/2-satisfied is possible. First, observe that R will only strictly contain RRs that are of even parity, as all RRs of odd parity were considered in step (1) of the algorithm and were either (a) transformed to RRs of even parity via merging or (b) expanded until their reached some

endpoint of the interval, and therefore can not be strictly contained in R. From this, it follows that every sub-interval of U_i that will be included in R on the left side of the cut on y will receive the same label (e.g., "+") and every sub-interval of U_i that will be included in R on the right side of the cut on y will receive the opposite label (e.g., "−"). □

Lemma 11. *After any step $j \geq i$, agent i is $1/2$-satisfied.*

Proof. First, it follows straightforwardly from Lemma 10 that agent i is $1/2$-satisfied after step i. For any $j > i$, let $R_i(k)$ be the union of RRs of I_i after step k and consider $R_i(i)$ and $R_i(j)$. By the way RRs are constructed, and since they are never intersected by a new cut, it follows that $R_i(i) \subseteq R_i(j)$ and in particular, the balance of labels in $R_i(i)$ has remained unchanged in $R_i(j)$ (although the labels themselves might have the opposite parity). It follows that the agent is $1/2$-satisfied after step j. □

The correctness of the algorithm follows by applying Lemma 11 for $j = n$.

Before we conclude the subsection, we mention that the algorithm can actually be applied to instances with piecewise constant valuations with d blocks, as long as we have $d \cdot n$ cuts at our disposal. The idea is quite simple: Any agent that has (at most) d value blocks will be replaced by (at most) d agents, with single-block valuations. This requires the appropriate scaling of the heights of the blocks, to make sure that the functions are still probability measures. Then, we will run the algorithm, now on $n' \leq d \cdot n$ agents, and we will obtain a solution using n' cuts. This set of cuts will also be a solution to the original instance, since the parameter ε is constant ($1/2$) in both cases. We obtain the following corollary.

Corollary 12. *There exists a polynomial-time algorithm for* $1/2$-Consensus-Halving, *when the agents have* piecewise constant *valuations with d* blocks *and we are allowed to use $d \cdot n$ cuts.*

4.3 A Polynomial time Algorithm using $2n - \ell$ cuts

We conclude the section with the following theorem, stating that there is a polynomial-time algorithm for the ε-Consensus-Halving problem with Single-block valuations, if we are allowed to use $2n - \ell$ cuts, for any constant ℓ.

Theorem 13. *Let ℓ be an integer constant. There exists a polynomial time algorithm that for any single block instance $\mathcal{C}$ of ε-Consensus-Halving computes a set of $2n - \ell$ cuts that define a solution. The running time of the algorithm is $O((2n)^{\ell}\text{poly}(n, \log(1/\varepsilon)))$.*

The algorithm is based on linear programming, and is presented below.

We will describe the algorithm for $t = 2n - 1$ cuts and then the extension to $2n - c$ cuts follows easily. Let $\vec{s} \in \mathbb{R}^t$ be a vector that contains set of possible positions of the cuts, where $s_i \geq s_j$ for any $i > j$ for simplicity we also set $s_0 = 0$ and $s_{2n} = 1$. Without loss of generality we also assume that we use the color $+$ in all the intervals of the form $[s_{2i}, s_{2i+1}]$ and we use the color $-$ in all the intervals of the form $[s_{2i+1}, s_{2(i+1)}]$. Assume also that all the measures are single block of the form $\mu_i(x) = \mathbf{1}\{x \in [a_i, b_i]\}$. Then for any given set of cuts $\vec{s}$ we have that the value of balance for every

measure i with respect to $\vec{s}$ is equal to

$$b_i(\vec{s}) = \sum_{j=1}^{n} \left[\min(z_{2j+1}, \max(b_i, z_{2j})) - \max(z_{2j}, \min(a_i, z_{2j+1}))\right] - \\ - \sum_{j=1}^{n} \left[\min(z_{2j+2}, \max(b_i, z_{2j+1})) - \max(z_{2j+1}, \min(a_i, z_{2j+2}))\right]. \quad (1)$$

Now we define the sequence $(c_k)_{k=1}^{2n}$ to be a sorted sequence of the set $\{a_j\}_{j=1}^{n} \cup \{b_j\}_{j=1}^{n}$. We split the interval $[0,1]$ into the sub-intervals of the form $[c_j, c_{j+1}]$ for all $j \in [2n-1]$. By the definition of the vector $\vec{c}$ it is clear that in any interval of the form $[c_j, c_{j+1}]$, no measure start and no measure ends. Because of this, we can observe that there is no need to put to cuts inside the same interval $[c_j, c_{j+1}]$. To see this assume that there exists a solution with two cuts $s_2 > s_1$ in $[c_j, c_{j+1}]$, then we can move the cuts to $s_1' = c_{j+1} - s_2 + s_1$ and $s_2' = c_{j+1}$ and nothing changes. Hence at most one cut in every interval $[c_j, c_{j+1}]$ is sufficient.

From our assumption we can use $2n-1$ cuts which is exactly equal to the number of intervals of the form $[c_j, c_{j+1}]$. We also define for simplicitly $c_0 = 0$ and $c_{2n} = 1$. Therefore we can restrict one cut s_j in every interval $[c_j, c_{j+1}]$, i.e. $c_j \leq s_j \leq c_{j+1}$. In this case all the terms that appear in (1) become linear since all the comparisons are fixed because of the aforementioned on the restriction of the cuts. Hence to find a solution that balances all the measures we need to solve the following satisfaction problem with linear constraints.

$$\begin{aligned} b_i(\vec{s}) &= 0 && \forall i \in [n] \\ c_j \leq s_j &\leq c_{j+1} && \forall j \in [2n-1] \end{aligned} \quad (2)$$

Finally if we do not have $2n-1$ cuts but $2n-\ell$ cuts instead, then we cannot assign one cut at every interval $[c_j, c_{j+1}]$ but we have to leave c of them with zero cuts. Since there is no clear way on how to choose which of them to leave out, we can try all the possible combinations which are bounded by $(2n-1)^{\ell}$, which is a polynomial in n for any fixed constant ℓ. For each one of these combinations we can check in polynomial time if the we can satisfy the resulting linear satisfaction problem until we find one solution. From these, Theorem 13 follows.

5 Consensus-1/3-Division is PPAD-hard

As we mentioned in the Introduction, our newly developed tools that allowed us to obtain a strengthening of the PPA-completeness result for Consensus-Halving, turn out to be very useful for proving a hardness result for a more general version of the problem, the Consensus-1/k-Division problem, for $k = 3$. We provide the definition below.

Definition 9 (ε-Consensus-1/k-Division)**.** Let $k \geq 2$. We are given $\varepsilon > 0$ and continuous probability measures $\mu_1, \ldots, \mu_n$ on $[0,1]$. The probability measures are given by their density functions on $[0,1]$. The goal is to partition the unit interval into k (not necessarily connected) pieces $A_1, \ldots, A_k$ using at most $(k-1)n$ cuts, such that $|\mu_j(A_i) - \mu_j(A_\ell)| \leq \varepsilon$ for all i, j, ℓ.

For ε-Consensus-1/3-Division, for ease of notation, we will use the labels A/B/C instead of 1/2/3. We state the main theorem of the section.

Theorem 14. *ε-Consensus-1/3-Division is PPAD-hard, for inverse-exponential ε.*

As we discussed in the Introduction, the problem for $k \geq 3$ is not necessarily harder than the case of $k = 2$, because we have more cuts at our disposal. Before we present the proof, we highlight some fundamental challenges that arise when one moves from $k = 2$ to larger k.

First, when moving to $k \geq 3$, we move from a simple $+/-$ parity to a more general setting with at least 3 labels. This is already severely problematic when it comes to the reduction of [Filos-Ratsikas and Goldberg, 2019], which is highly dependent on the solution having two labels. Indeed, the interpretation of the inputs in [Filos-Ratsikas and Goldberg, 2019] and the corresponding design of the gates needs to know which label will appear on the left side of the cut, and special "parity-flip" gadgets are used throughout the reduction to ensure this. On the contrary, with our new value interpretation, we design gadgets which take the label sequence "internally" into account, by adjusting the position of the cut accordingly.

The sequence of the labels however gives rise to a second and more challenging issue: While in the case of $k = 2$ we can assume without loss of generality that the labels between cuts alternate between "+" and "−", we cannot make any such assumptions even when $k = 3$. In fact, it is known that if one restricts the solution to exhibit a cyclic sequence of labels $A/B/C$, then the problem is no longer a total search problem [Pálvölgyi, 2009]. This seems to be a fundamental obstacle to the design of gates for the case of $k \geq 3$. For $k = 3$, we manage to side-step this obstacle by using a clever "trick": we make sure that the intervals of the Consensus-1/3-Division instance where we read the two inputs (of the 2-dimensional PPAD-complete problem that we reduce from, see Definition 11) are placed next to each other, therefore fixing the position of one of the three labels. We prove PPAD-hardness for the *exact* version of the problem (in which we are looking for a perfect balance of the labels, with no allowable discrepancy ε), which will guarantee that in a solution, the value of this label will be fixed to $1/3$ throughout the instance. Since our instance is constructed to have piecewise constant valuation functions, the result can be extended to the case of inverse-exponential ε using the following lemma, which is based on an argument of Etessami and Yannakakis [2010].

Lemma 15. *Let $k \geq 2$. For piece-wise constant valuation functions, exact* Consensus-1/k-Division *reduces in polynomial time to* ε-Consensus-1/k-Division *with inverse-exponential ε.*

Proof. We use the proof idea introduced by Etessami and Yannakakis [2010]. Given an instance of Consensus-1/k-Division with piecewise constant valuations that we want to solve exactly, we first find an ε-approximate solution S for some sufficiently small ε. Then, using S, we construct an LP in polynomial time. Finally, we show that any solution to this LP yields an exact solution to the Consensus-1/k-Division instance.

Let I be an instance of Consensus-1/k-Division with agents $\{1, 2, \ldots, n\}$ that have piecewise constant valuations. Then, we can efficiently find positions $0 = p_0 < p_1 < \cdots < p_{m-1} < p_m = 1$ such that for all agents $i \in [n]$ and all $j \in [m]$, the density of the valuation function μ_i in interval $[p_{j-1}, p_j]$ is constant. Denote that constant by $f_{i,j}$. Note that m and the bit-length of $f_{i,j}$ are polynomially upper bounded by the size of the instance I.

Let $S = (\widehat{x}, \text{label})$ be a candidate solution, where the cut positions are $0 = \widehat{x}_0 \leq \widehat{x}_1 \leq \widehat{x}_2 \leq \cdots \leq \widehat{x}_{(k-1)n-1} \leq \widehat{x}_{(k-1)n} \leq \widehat{x}_{(k-1)n+1} = 1$ and for $t \in [(k-1)n+1]$, label(t) is the label in $[k]$ assigned to interval $[\widehat{x}_{t-1}, \widehat{x}_t]$.

We construct an LP that has variables $x_0, \ldots, x_{(k-1)n+1}$ and a variable z. We denote it by LP(I, S). In the LP we minimize z under the constraints:

- $x_0 = 0$ and $x_{(k-1)n+1} = 1$ (these variables are only used for notational convenience)
- for every $t \in [(k-1)n]$: cut x_t must lie between cuts x_{t-1} and x_{t+1}

- for every $t \in [(k-1)n]$: for every j such that $\widehat{x}_t \in [p_{j-1}, p_j]$ (at most two such j exist), x_t must also lie in $[p_{j-1}, p_j]$
- let $A_1, \ldots, A_k$ denote the partition of the domain $[0,1]$ given by the cuts $x_0, \ldots, x_{(k-1)n+1}$ and the labeling label. Then the constraint is:

  for every agent $i \in [n]$: $\max_{\ell_1,\ell_2} |\mu_i(A_{\ell_1}) - \mu_i(A_{\ell_2})| \leq z$

We could also add a constraint $z \geq 0$, but this is already implicitly enforced by the existing constraints.

Apart from the last constraint type, it is clear that all the other constraints can be expressed linearly. For the last type of constraint, note that it can be broken down into all the constraints $\mu_i(A_{\ell_1}) - \mu_i(A_{\ell_2}) \leq z$ and $\mu_i(A_{\ell_2}) - \mu_i(A_{\ell_1}) \leq z$ for all ℓ_1, ℓ_2. Thus, it remains to show that $\mu_i(A_{\ell_1})$ and $\mu_i(A_{\ell_2})$ can be written as linear functions of $x_0, \ldots, x_{(k-1)n+1}$.

To see this, first note that the labeling is fixed, i.e. for any interval $[x_t, x_{t+1}]$ we know which label it is assigned to. Furthermore, for every $t \in [(k-1)n]$ the cut x_t is constrained to lie in some interval $[p_{j-1}, p_j]$, and it is not allowed to cross from one interval $[p_{j-1}, p_j]$ to another. Thus, for any agent $i \in [n]$ and for any interval $[p_{j-1}, p_j]$, we can express the amount of value of agent i in $[p_{j-1}, p_j]$ going to each of the labels $\{1, 2, \ldots, k\}$ as a linear expression. If $[p_{j-1}, p_j]$ does not contain any cut x_t, then all of it is assigned to some label ℓ (which is fixed, i.e. only depends on S). Thus, the value assigned to label ℓ is $f_{ij}(p_j - p_{j-1})$, and the value assigned to all other labels is 0. If $[p_{j-1}, p_j]$ contains the cuts $x_{t_1} \leq \cdots \leq x_{t_s}$, then:

- interval $[p_{j-1}, x_{t_1}]$ yields value $f_{ij}(x_{t_1} - p_{j-1})$ to label(t_1) (and value 0 to all other labels)
- interval $[x_{t_p}, x_{t_{p+1}}]$ (for $1 \leq p \leq s-1$) yields value $f_{ij}(x_{t_{p+1}} - x_{t_p})$ to label(t_{p+1})
- interval $[x_{t_s}, p_j]$ yields value $f_{ij}(p_j - x_{t_s})$ to label$(t_s + 1)$

Note that any feasible solution to this LP with $z = 0$ is an exact solution to our Consensus-$1/k$-Division instance I. Furthermore, note that there exist polynomials p_1 and p_2 such that for any candidate solution S, the number of equations in LP(I, S) is at most $p_1(|I|)$ and the bit-size of any number appearing in LP(I, S) is at most $p_2(|I|)$. Here $|I|$ denotes the representation size of the input I. Thus, there exists some polynomial q such that we can efficiently compute an optimal solution of LP(I, S) where the bit-size of the variables is at most $q(|I|)$. Note that this does not depend on S.

Now assume that we have picked $\varepsilon < 1/2^{q(|I|)}$ and obtain a solution $S = (\widehat{x}, \text{label})$ to ε-Consensus-$1/k$-Division(I). We then construct LP(I, S) and compute an optimal solution (x^*, z^*). Note that $(\widehat{x}, \varepsilon)$ is a feasible solution to LP(I, S). Thus, it must hold that $z^* \leq \varepsilon < 1/2^{q(|I|)}$. But since the bit-size of z^* is at most $q(|I|)$ and $z^* \geq 0$, it must hold that $z^* = 0$. Thus, $S' = (x^*, \text{label})$ is an exact solution to our Consensus-$1/k$-Division instance I. □

5.1 A Problem to Reduce From: 2D-Truncated-Linear-FIXP

We start with the following problem, proven to be PPAD-complete by Mehta [2014].

Definition 10 (2D-Linear-FIXP [Mehta, 2014])**.** The problem 2D-Linear-FIXP is defined as follows. We are given a circuit C using gates $\{+, \times\zeta, \max\}$ and rational constants, that computes a function $F_C : [0,1]^2 \to [0,1]^2$. The goal is to find $x \in [0,1]^2$ such that $F_C(x) = x$.

Theorem 16 ([Mehta, 2014])**.** 2D-Linear-FIXP *is* PPAD*-complete.*

In order to prove PPAD-hardness of Consensus-1/3-Division, we will use a slightly modified version of 2D-Linear-FIXP, that we call 2D-Truncated-Linear-FIXP. The first difference is that the domain is $[-1,1]^2$ instead of $[0,1]^2$. Furthermore, instead of the gates $\{+, \times\zeta, \max\}$ and rational constants in $\mathbb{Q}$, the circuit will only be allowed to use the gates $\{+_T, \times_T\zeta\}$ and rational constants in $[-1,1] \cap \mathbb{Q}$. The gate $+_T$ corresponds to *truncated addition* and the gate $\times_T\zeta$ corresponds to *truncated multiplication by* ζ. For any $x, y \in [-1,1]$ and any $\zeta \in \mathbb{Q}$, we have $x +_T y = T[x+y]$ and $x \times_T \zeta = T[x \times \zeta]$, where T is the truncation operator in $[-1,1]$ as defined in Section 3.

Definition 11 (2D-Truncated-Linear-FIXP)**.** The problem 2D-Truncated-Linear-FIXP is defined as follows. We are given a circuit C using gates $\{+_T, \times_T\zeta\}$ and rational constants in $[-1,1]$, that computes a function $F_C : [-1,1]^2 \to [-1,1]^2$. The goal is to find $x \in [-1,1]^2$ such that $F_C(x) = x$.

By applying some simple modifications to any 2D-Linear-FIXP circuit, we are able to show the following theorem.

Theorem 17. 2D-Truncated-Linear-FIXP *is* PPAD*-complete.*

Proof. Containment in PPAD follows from the containment of the more general problem Linear-FIXP in PPAD [Etessami and Yannakakis, 2010]. In particular, note that the truncated gates can be simulated by using their non-truncated versions, along with max and constant-gates.

To show PPAD-hardness, we reduce from 2D-Linear-FIXP. Let C be a circuit that computes a function $F_C : [0,1]^2 \to [0,1]^2$ that uses gates $\{+, \max, \times\zeta\}$ and rational constants. First, let us change the domain to $[-1,1]^2$. We modify the circuit C into $\widehat{C}$ by applying the following transformation (also used in [Mehta, 2014]) to every output: $x \mapsto \max\{\min\{1,x\},0\} = \max\{-1 \times \max\{-1, -1 \times x\}, 0\}$. Then, for any input $(x,y) \in [-1,1]^2$, we must have that $F_{\widehat{C}}(x,y) \in [0,1]^2$ and for $(x,y) \in [0,1]^2$ we have $F_{\widehat{C}}(x,y) = F_C(x,y)$. It follows that $\widehat{C}$ computes a function $F_{\widehat{C}} : [-1,1]^2 \to [-1,1]^2$ and any fixed-point of $F_{\widehat{C}}$ must be in $[0,1]^2$ and thus also a fixed-point of F_C. This means that without loss of generality, we can assume that the domain is $[-1,1]^2$ instead of $[0,1]^2$.

Next, let us show how to replace the gates $\{+, \times\zeta\}$ by $\{+_T, \times_T\zeta\}$, and ensure that the constant-gates all have value in $[-1,1]$. Let $c \geq 2$ be an upper bound on the absolute value of all the constants appearing in the circuit, i.e. both the constant-gates and the $\times\zeta$-gates. Note that c has bit representation length that is polynomial in the size of C (since the size of C includes the representation length of all constants). Let n be the number of gates $\{+, \max, \times\zeta\}$ in C.

Begin with the circuit C_0 that only contains the two input gates of C and all the constant-gates of C. Pick an arbitrary gate in C with input(s) in C_0 and add it to C_0 to obtain circuit C_1. Then, pick an arbitrary gate in C (and not yet in C_1) with input(s) in C_1 and add it to C_1 to obtain C_2. If we repeat this n, we obtain an incremental sequence of circuits $C_0, C_1, \ldots, C_n = C$, where a single gate is added at each step.

For $i \in \{0, 1, \ldots, n\}$, let $v(C_i)$ denote the maximum absolute value of any gate in C_i, over all inputs in $[-1,1]^2$. Clearly, we have $v(C_0) \leq \max\{1, c\} = c$. To obtain C_1 from C_0, we have added a single gate. If this gate is a $+$-gate, then maximum absolute value appearing in the circuit is at most multiplied by $2 \leq c$. If it is a max-gate, then the maximum absolute remains the same. Finally, if it is a $\times\zeta$-gate, the maximum absolute value is at most multiplied by $|\zeta| \leq c$. Thus, in any case, we have $v(C_0) \leq cv(C_1)$. By induction it follows that $v(C) = v(C_n) \leq c^n v(C_0) = c^{n+1}$. Note that $M := c^{n+1}$ has representation length that is polynomial with respect to the size of C.

From the arguments above, it follows that if the input is in $[-1,1]^2$, all the intermediate values in the computation of the circuit are upper bounded by M in absolute value. Now modify C to obtain C' as follows. For every constant-gate, replace its constant ζ by ζ/M. For every input,

introduce a $\times(1/M)$-gate that multiplies it by $1/M$ and then use the output of that gate as the corresponding input for C. By induction, it is easy to see that on any input $(x, y) \in [-1, 1]^2$, C' computes $F_C(x, y)/M$. Importantly, all the intermediate values in the computation of the circuit are now upper bounded by 1 in absolute value. In particular, all the constant-gates have value in $[-1, 1]$.

For any $(x, y) \in [-1, 1]^2$, we know that $F_C(x, y) \in [-1, 1]^2$, i.e. $F_C(x, y)/M \in [-1/M, 1/M]^2$. Obtain circuit C'' by taking C' and multiplying all the outputs by M (by using $\times M$-gates). Then, on any input $(x, y) \in [-1, 1]^2$, C'' outputs $F_C(x, y)$ and all intermediate values in the computation of the circuit are upper bounded by 1 in absolute value.

Thus, we have transformed C into a circuit that computes the same function $F_C : [-1, 1]^2 \to [-1, 1]^2$, but only uses gates $\{+_T, \times_T\zeta, \max\}$, and all the constant-gates have value in $[-1, 1]$.

The last step is to get rid of max-gates. To do this observe that for any $x, y \in [-1, 1]$, we have

$$\max\{x, y\} = \left(\frac{1}{2} \times_T x + \max\left\{\frac{1}{2} \times_T y +_T \left(-\frac{1}{2}\right) \times_T x, 0\right\}\right) \times_T 2$$

and

$$\max\{x, 0\} = (x +_T (-1)) +_T 1$$

Using these two equations we can implement any max-gate by using the gates $\{+_T, \times_T\zeta\}$ and rational constants in $[-1, 1]$.

Putting everything together, we have provided a reduction from 2D-Linear-FIXP to 2D-Truncated-Linear-FIXP. □

5.2 Description of the Reduction

In order to show that Consensus-1/3-Division is PPAD-hard, we reduce from 2D-Truncated-Linear-FIXP. Namely, given a 2D-Truncated-Linear-FIXP circuit C, we will construct an instance I_C of Consensus-1/3-Division, such that any solution of I_C yields a solution to C (i.e., a fixed-point of $F_C : [-1, 1]^2 \to [-1, 1]^2$). As before, we will construct a Consensus-1/3-Division instance on some domain $[0, M]$ for some polynomial M, which is easy to transform into an equivalent instance on domain $[0, 1]$.

First, let us give a high-level description of the *ideal* reduction that we would like to construct. First, we would show how any interval of the Consensus-1/3-Division domain encodes a value in $[-1, 1]$. Namely, in any solution S to instance I_C, for every interval I, $v_S(I) \in [-1, 1]$ would be the value encoded by interval I. Then, we would construct agents that implement the arithmetic gates needed by 2D-Truncated-Linear-FIXP, namely $\{+_T, \times_T\zeta\}$ and rational constants in $[-1, 1]$. These agents read some value(s) in $[-1, 1]$ from one or two intervals and output the result of the gate-operation into some other interval of the domain.

With these gates we could implement the circuit C inside our Consensus-1/3-Division instance. In particular, we would have two intervals In_1 and In_2 each representing the two inputs, and two intervals Out_1 and Out_2 each representing the two outputs. These intervals are pairwise disjoint. In the final step we would then "connect" the outputs to the inputs. Namely, we would introduce an agent implementing a $\times_T 1$-gate with input Out_1 and output In_1, and a second agent implementing a $\times_T 1$-gate with input Out_2 and output In_2. This ensures that from any solution of the Consensus-1/3-Division instance we can extract a fixed-point of F_C.

If we could do all this, then this reduction would be very similar to the reduction of Filos-Ratsikas et al. [2018] showing that ε-Consensus-Halving is PPAD-hard for constant ε. Unfortunately, there is a significant obstacle. Namely, we don't know how to find an encoding of values in

intervals such that we can implement arithmetic gates that always work. Because we don't know in what order the labels A, B and C will appear in any given interval, implementing arithmetic gates is actually much harder than in the case of Consensus-Halving. Thus, the gates we are able to implement only work if the input interval encodes a value in a very specific way. In this case, we say that the interval is a *valid encoding* of a value. Not all intervals will be a valid encoding of a value. In general, it is very hard to enforce valid encodings. This is the reason why our reduction does not seem to generalize to yield hardness for inversely polynomial ε, or to Consensus-$1/k$-Division with $k > 3$.

Nevertheless, for exact Consensus-1/3-Division we are able to find a work-around to force all intervals to be valid encodings of a value. If an interval is a valid encoding of a value (in solution S), then let $v_S(I) \in [-1, 1]$ denote the value encoded by I. We will drop the subscript S in the remainder of the exposition. The following two lemmas are crucial. They are proved in Section 5.3, where the proof of Theorem 14 is detailed.

Lemma 18. *In the instance I_C we construct, it holds that:*

- *the two intervals In_1 and In_2 are valid encodings, and*
- *if Out_1 and Out_2 are valid encodings, then $v(Out_1) = v(In_1)$ and $v(Out_2) = v(In_2)$.*

Lemma 19. *In instance I_C, we can implement arithmetic gates $\{G_{+_T}, G_{\times_T\zeta}, G_\zeta\}$ for operations $\{+_T, \times_T\zeta\}$ and constant-gate respectively, such that:*

- *G_ζ outputs a valid encoding of $\zeta \in [-1, 1] \cap \mathbb{Q}$*
- *if the input to $G_{\times_T\zeta}$ is a valid encoding of value $x \in [-1, 1]$, then the gate outputs a valid encoding of $x \times_T \zeta$*
- *if the two inputs to G_{+_T} are valid encodings of $x, y \in [-1, 1]$, then the gate outputs a valid encoding of $x +_T y$.*

If these two lemmas indeed hold, then the reduction is correct. First of all, by Lemma 18, the two inputs to the circuit are valid encodings. Thus, using Lemma 19, it follows by induction that all the gates in the circuit perform their operation correctly and output a valid encoding. In particular, the two outputs of the circuit are valid encodings. Thus, by Lemma 18, we get that each of the two outputs is equal to the corresponding input. As a result, we have identified a fixed-point of $F_C : [-1, 1]^2 \to [-1, 1]^2$.

5.3 Details of the Proof

We construct an instance with a set of agents $\{1, 2, \ldots, n\}$ for some n polynomial in the size of C. For every $i \in [n]$, agent i will have an *output interval* O_i of length 9 on the domain with the following properties:

- for all $j \neq i$ it holds $O_j \cap O_i = \emptyset$ (output intervals are pairwise disjoint)
- agent i's valuation function has density $3/10$ in $O_i[1, 2] \cup O_i[4, 5] \cup O_i[7, 8]$.

From this it follows that in any solution S, for each $i \in [n]$, O_i must contain at least two cuts. Since the intervals O_i are pairwise disjoint, all $2n$ cuts are accounted for and thus for each $i \in [n]$, O_i contains exactly two cuts. But then it follows that O_i must be *well-cut*: one cut lies in $O_i[7/4, 17/4]$ and the other one in $O_i[19/4, 29/4]$. In particular, there are no stray cuts in this reduction.

Representation of a value. In any solution S, every interval I encodes a value as follows. Let $X(I) = I[0,1] \cup I[2,4] \cup I[5,7] \cup I[8,9]$. Let $X(I)_A$, $X(I)_B$ and $X(I)_C$ denote the subsets of $X(I)$ allocated by the solution S to labels A, B and C respectively. Let μ denote the Lebesgue measure on $\mathbb{R}$. We say that I is a *valid encoding* of a value, if $\mu(X(I)_C) = 2$ and if I is well-cut. If I is a valid encoding, then the encoded value is $v(I) = (\mu(X(I)_A) - \mu(X(I)_B))/2 \in [-1,1]$.

In the next section, we show how to construct the arithmetic gates. With these gates, we can implement the circuit C in our instance I_C. We will implement the circuit such that the first output of the circuit is encoded in the left-most interval of the domain $Out_1 = [0,9]$, and the second output of the circuit is encoded in the next interval, i.e. $Out_2 = [10,19]$. We will ensure that the next two intervals are not used by any gate of the circuit, namely $Temp_1 = [20,29]$ and $Temp_2 = [30,39]$ are available. Finally, the next two intervals correspond to the two inputs of the circuit, i.e. $In_1 = [40,49]$ and $In_2 = [50,59]$.

By construction, we know that Out_1 is well-cut, because it is the output interval of some agent. In particular, it contains exactly two cuts. By convention, we will assume that the labels appear in order A, B, C from left to right in Out_1. Given any solution S of I_C, we can always ensure that this holds by renaming the labels. Since Out_1 is well-cut, we know that interval $Out_1[0,1/2]$ is labeled A, interval $Out_1[17/4,19/4]$ is labeled B, and interval $Out_1[17/2,9]$ is labeled C. Furthermore, since interval Out_2 is also well-cut and right next to Out_1, we know that $Out_2[0,1/2]$ is labeled C. However, we do not know the labels of $Out_2[17/4,19/4]$ and $Out_2[17/2,9]$, except that one of them is A and the other B (but not which one is which).

Projection Agents. In order to ensure that Lemma 18 holds, we are going to introduce two special agents, that we call *projection agents*. Let $I := Out_1$ and $O := Temp_1$. The first projection agent's valuation function has height $3/10$ in $O[1,2] \cup O[4,5] \cup O[7,8]$, height $1/120$ in $X(O)$, height $1/30$ in $I[17/2,9]$, height $1/120$ in $I[2,4]$, height $1/60$ in $I[0,1/2] \cup I[17/4,19/4]$, and height 0 everywhere else. Since Out_1 is well-cut and we know the labels of the pieces, this agent ensures that:

- interval $Temp_1$ is a valid encoding
- if interval Out_1 is a valid encoding, then $v(Temp_1) = -v(Out_1)$

The first property holds because exactly $1/3$ of the projection agent's value in interval Out_1 goes to label C. Thus, label C also needs to get exactly $1/3$ of the agent's value in interval $Temp_1$. By construction of the valuation function in $Temp_1$, it follows that C must get exactly $1/3$ of $X(Temp_1)$. To show the second property, consider the three possible cases:

- label C is the right-most label in $Temp_1$: then there is a cut at position $Temp_1[6]$. In this case, it is easy to see that the other cut in $Temp_1$ will exactly reflect what the first cut in Out_1 does.
- label C is the middle label in $Temp_1$: since C gets $1/3$ of $X(Temp_1)$, it follows that the cuts in $Temp_1$ have distance exactly 3 between them. From this, it follows that both cuts reflect what the first cut in Out_1 does.
- label C is the left-most label in $Temp_1$: then there is a cut at position $Temp_1[3]$. Again, as in the first case, the other cut in $Temp_1$ will exactly reflect what the first cut in Out_1 does. (Note that this case is actually not possible here, but we have included it, because it might occur for the second projection agent).

In order to show that Lemma 18 holds for In_1 and Out_1, it suffices to use a $G_{\times_T -1}$-gate (see next section) with input $Temp_1$ and output In_1.

Now let $I := Out_2$ and $O := Temp_2$. The second projection agent's valuation function has height $3/10$ in $O[1,2] \cup O[4,5] \cup O[7,8]$, height $1/120$ in $X(O)$, height $1/30$ in $I[0,1/2]$, height $1/120$ in $I[5,7]$, height $1/60$ in $I[17/4,19/4] \cup I[17/2,9]$, and height 0 everywhere else. In other words, the second projection agent's valuation function in interval Out_2 is the same as the first projection agent's in Out_1, except that it is mirrored. By using the same arguments we used for the first projection agent, we get that Lemma 18 also holds for In_2 and Out_2.

5.3.1 Arithmetic Gates

In this section, we prove Lemma 19, by providing an explicit construction of the arithmetic gates. As before, let $T : \mathbb{R} \to [-1,1]$ denote the truncation operator. When constructing the agents implementing the gates, we can assume that their output interval is well-cut, since this holds for all agents by construction.

Multiplication by a constant $\zeta \in \mathbb{Q}$ $[G_{\times_T \zeta}]$: Let I and O be two disjoint intervals of length 9. First consider the case $\zeta \leq 0$. We create an agent with the following valuation function. The agent's density function has height $3/10$ in $O[1,2] \cup O[4,5] \cup O[7,8]$, height $\frac{|\zeta|}{60(|\zeta|+1)}$ in $X(I)$, height $\frac{1}{60(|\zeta|+1)}$ in $X(O)$, and height 0 everywhere else. If I is a valid encoding, then O is a valid encoding of the value $v(O) = T[v(I) \times \zeta]$. If $\zeta > 0$, then use a $G_{\times_T -\zeta}$-gate and then a $G_{\times_T -1}$-gate.

Let us see why this works. Since I encodes a valid value, exactly $1/3$ of the agent's value in interval I goes to label C. It follows that exactly $1/3$ of the agent's value in interval O must go to label C. By construction of the valuation function in O, it follows that $1/3$ of $X(O)$ must go to label C. As a result, O is also a valid encoding. Now, if label C gets the left-most piece in O, then there is a cut at position $O[3]$. The other cut, which is located in $O[19/4, 29/4]$, will thus encode the value of interval O and the truncation will work similarly to our Consensus-Halving gadgets. The analogous argument also holds if C gets the right-most piece in O. If label C gets the middle piece in O, then the distance between the two cuts will be exactly 3. Thus, the two cuts "move together" and will touch a big block at the same time. As a result, the truncation works here as well.

Addition $[G_{+_T}]$: Let I_1, I_2 and O be three pairwise disjoint intervals of length 9. The agent's density function has height $3/10$ in $O[1,2] \cup O[4,5] \cup O[7,8]$, height $1/180$ in $X(I_1) \cup X(I_2) \cup X(O)$, and height 0 everywhere else. If I_1 and I_2 are valid encodings, then O is a valid encoding of the value $v(O) = -T[v(I_1) + v(I_2)]$. To obtain G_{+_T}, just apply a $G_{\times_T -1}$-gate on the output. A similar reasoning to the case of $G_{\times_T \zeta}$ proves the correctness of this gate too.

Constant $\zeta \in [-1,1] \cap \mathbb{Q}$ $[G_\zeta]$: For this we use the left-most interval of length 9 of the domain, namely Out_1. We know that interval $I := Out_1$ is the output interval of some gate, so it is well-cut. Furthermore, we know that the labels appear in order A, B, C from left to right (by convention). Thus, we know that interval $I[0,1/2]$ is labeled A, interval $I[17/4,19/4]$ is labeled B, and interval $I[17/2,9]$ is labeled C.

Introduce an agent with output interval O of length 9. The agent's density function has height $3/10$ in $O[1,2] \cup O[4,5] \cup O[7,8]$, height $1/120$ in $X(O)$, height $1/30$ in $I[17/2,9]$, height $\frac{1-\zeta/2}{30}$ in

$I[0, 1/2]$, height $\frac{1+\zeta/2}{30}$ in $I[17/4, 19/4]$, and height 0 everywhere else. From this construction, it follows that interval O is a valid encoding of the value $v(O) = \zeta$.

6 Future Directions

The main technical question is whether ε-Consensus-Halving for single-block valuations is PPA-hard or polynomially solvable, or perhaps even complete for some other class (like PPAD or CLS). Another interesting direction is to extend the PPA-hardness result of Theorem 2 (or even for a larger number of blocks) to constant ε; such a result however would seemingly require some radically new ideas, namely an averaging argument over a constant set of outputs that is robust to stray cuts. Finally, it would be interesting to study the complexity of the Consensus-$1/k$-Division problem when $k \geq 3$ and possibly strengthen or extend our hardness result to other values of k.

Acknowledgments

Alexandros Hollender is supported by an EPSRC doctoral studentship (Reference 1892947). Katerina Sotiraki is supported in part by NSF/BSF grant #1350619, an MIT-IBM grant, and a DARPA Young Faculty Award, MIT Lincoln Laboratories and Analog Devices. Part of this work was done while the author was visiting the Simons Institute for the Theory of Computing. Manolis Zampetakis is supported by a Google Ph.D. Fellowship and NSF Award #1617730.

We would like to thank Paul Goldberg for helpful discussions and comments.

References

James Aisenberg, Maria Luisa Bonet, and Sam Buss. 2-D Tucker is PPA complete. *Journal of Computer and System Sciences*, 108:92–103, 2020.

Noga Alon. Splitting necklaces. *Advances in Mathematics*, 63(3):247–253, 1987.

Noga Alon and Douglas B. West. The Borsuk-Ulam Theorem and Bisection of Necklaces. *Proceedings of the American Mathematical Society*, 1986. ISSN 00029939. doi: 10.2307/2045739.

Eshwar Ram Arunachaleswaran, Siddharth Barman, Rachitesh Kumar, and Nidhi Rathi. Fair and efficient cake division with connected pieces. In *Proceedings of the 15th Conference on Web and Internet Economics (WINE)*, pages 57–70. Springer, 2019.

Haris Aziz and Simon Mackenzie. A discrete and bounded envy-free cake cutting protocol for any number of agents. In *Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 416–427. IEEE, 2016a.

Haris Aziz and Simon Mackenzie. A discrete and bounded envy-free cake cutting protocol for four agents. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 454–464, 2016b.

Imre Bárány, Senya B. Shlosman, and András Szücs. On a topological generalization of a theorem of Tverberg. *Journal of the London Mathematical Society*, 2(1):158–164, 1981.

Karol Borsuk. Drei Sätze über die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 1933. ISSN 0016-2736. doi: 10.4064/fm-20-1-177-190.

Steven J Brams and Alan D Taylor. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

Xi Chen, Xiaotie Deng, and Shang-Hua Teng. Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, 56(3):14, 2009.

Xi Chen, Dimitris Paparas, and Mihalis Yannakakis. The complexity of non-monotone markets. In *Proceedings of the 45th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 181–190, 2013.

Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a Nash equilibrium. In *SIAM Journal on Computing*, volume 39, pages 195–259, 2009. doi: 10.1137/070699652.

Argyrios Deligkas, John Fearnley, Themistoklis Melissourgos, and Paul G. Spirakis. Computing Exact Solutions of Consensus Halving and the Borsuk-Ulam Theorem. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 138:1–138:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Xiaotie Deng, Qi Qi, and Amin Saberi. Algorithmic solutions for envy-free cake cutting. *Operations Research*, 60(6):1461–1476, 2012.

Xiaotie Deng, Jack R. Edmonds, Zhe Feng, Zhengyang Liu, Qi Qi, and Zeying Xu. Understanding PPA-completeness. In *Proceedings of the 31st Conference on Computational Complexity (CCC)*, pages 23:1–23:25. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

Xiaotie Deng, Zhe Feng, and Rucha Kulkarni. Octahedral Tucker is PPA-Complete. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 24, page 118, 2017.

Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

Aris Filos-Ratsikas and Paul W. Goldberg. Consensus Halving is PPA-complete. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 51–64. ACM, 2018.

Aris Filos-Ratsikas and Paul W. Goldberg. The Complexity of Splitting Necklaces and Bisecting Ham Sandwiches. In *Proceedings of the 51st Annual ACM Symposium on Theory of Computing (STOC)*, pages 638–649. ACM, 2019.

Aris Filos-Ratsikas, Søren Kristoffer Still Frederiksen, Paul W. Goldberg, and Jie Zhang. Hardness Results for Consensus-Halving. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 24:1–24:16. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

Jugal Garg, Ruta Mehta, and Vijay V. Vazirani. Substitution with satiation: A new class of utility functions and a complementary pivot algorithm. *Mathematics of Operations Research*, 43(3): 996–1024, 2018.

Charles H. Goldberg and Douglas B. West. Bisection of Circle Colorings. *SIAM Journal on Algebraic Discrete Methods*, 1985. ISSN 0196-5212. doi: 10.1137/0606010.

Paul W. Goldberg and Alexandros Hollender. The Hairy Ball Problem is PPAD-Complete. In *Proceedings of the 46th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 65:1–65:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Mika Göös, Pritish Kamath, Katerina Sotiraki, and Manolis Zampetakis. On the Complexity of Modulo-q Arguments and the Chevalley-Warning Theorem. *arXiv preprint arXiv:1912.04467*, 2019.

Michelangelo Grigni. A Sperner lemma complete for PPA. *Information Processing Letters*, 2001. ISSN 00200190. doi: 10.1016/S0020-0190(00)00152-6.

Charles R. Hobby and John R. Rice. A moment problem in L1 approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965.

Alexandros Hollender. The Classes PPA-k: Existence from Arguments Modulo k. In *Proceedings of the 15th Conference on Web and Internet Economics (WINE)*, pages 214–227. Springer, 2019.

Nimrod Megiddo and Christos H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 1991. ISSN 03043975. doi: 10.1016/0304-3975(91)90200-L.

Ruta Mehta. Constant rank bimatrix games are PPAD-hard. In *Proceedings of the 46th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 545–554, 2014.

Jerzy Neyman. Un theoreme d'existence. *CR Acad. Sci. Paris*, 222:843–845, 1946.

Dömötör Pálvölgyi. Combinatorial Necklace Splitting. *The Electronic Journal of Combinatorics*, 16(1) (R79):1–8, 2009.

Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and System Sciences*, 48(3):498–532, 1994.

Forest W. Simmons and Francis Edward Su. Consensus-halving via theorems of Borsuk-Ulam and Tucker. *Mathematical social sciences*, 45(1):15–25, 2003.

Hugo Steinhaus. The Problem of Fair Division. *Econometrica*, 16:101–104, 1948.

Albert W. Tucker. Some Topological Properties of Disk and Sphere. In *Proceedings of the First Canadian Math. Congress, Montreal*, pages 286–309. University of Toronto Press, 1945.